CM FIRST
Rethink Modernization

CM WebClient

Manual v 1.8.2

CM WebClient

This documentation and related computer software program (hereinafter referred to as "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by CM1 Technologies, Inc. d.b.a. CM First Technologies ("CM1") at any time.

Documentation may not be copied, transferred, reproduced, disclosed, or duplicated, in whole or in part, without the prior written consent of CM1. Documentation is proprietary information of CM1 and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of Documentation for their own internal use, provided that all CM1 copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

To the extent permitted by applicable law, CM1 provides Documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or non-infringement. In no event will CM1 be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of Documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CM1 is advised of the possibility of such loss or damage.

The use of any product referenced in Documentation and Documentation is governed by the end user's applicable license agreements.

The manufacturer of Documentation is CM First Technologies.

© 2013 CM First Technologies

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Table of Contents

1. Introduction 6
   - What's New in CM WebClient 1.8.2 6
   - Current Limitations 8
   - Using the Live Installer 8
2. Installation Requirements 9
3. Plex Model Configuration 10
   - Overview 10
   - Instructions 10
   - Plex v6.1/Plex v7.0 Model Configuration 11
4. Overview of Workspace Setup 12
   - Install WebClient Builder Plugin 13
   - Configure Apache Tomcat in Eclipse 14
   - Create a New Java Project 15
   - Import PlexBuild project 17
   - Setup PlexBuild Run Configuration 18
   - Configure build.properties File in PlexBuild project 23
   - How to Run PlexBuild in Eclipse 23
   - Import WebClient project 24
   - Configure Java Project 25
   - Configure obclient.properties 26
   - Generate WebClient Templates 27
   - Create a New Dynamic Web Project 29
   - Configure Build Path for the New Web Project 31
   - Create a New Servlet in Web Project 32
   - Create a New Listener in WebClient Servlet 34
   - Add Plex Images to the Web Project 35
   - Run WebClient in Eclipse 35
   - Configure WebClient.properties 37
5. Deploying CM WebClient with Websydian Express for Java 40
   - Introduction 40
   - Instructions 40
   - Complete Steps 3 – 11 of Section 4 40
   - Import the Express30.war File 41
   - Configure J2eeProxy.properties 41

Add WebClient.properties to Express30 Project 42
Configure WebClient.properties. 42
Configure obusr.properties. 42
Configure express30 project. 43
Configure Websydian Express Admin site. 44
Run CM WebClient application 45
6. Calling CM WebClient from an External Application 46
7. Configuring Session Timeout in Apache Tomcat 50
8. Upgrading CM WebClient v1.6.x to CM WebClient v1.8.x 51
9. Customizing in CM WebClient 53
Attach point 53
Bean controls 54
Control action 54
Control context 55
Control name property 55
Control reference 56
Control template 56
Default attach points for controls 57
Name identifier 57
Output path 57
Page template 58
Physical event 58
Property modifier 59
Root template 59
Source 60
Template generation 60
10. Servlet 3.0 Template Resource Deployment 61
Introduction 61
Web Library ID 61
11. How to Customize CM WebClient StyleSheet 62
12. Troubleshooting 63
Appendix A: Template Commands 65
Appendix B: JDBC Driver Configuration 80
Configuring and Using SQL Server 2005 80
Creating an ODBC Connection for SQL Server 2005 80
Configuring the SQL Server 2005 JDBC Driver in Eclipse 84

Configuring and Using MySQL Server 87
Configuring MySQL JDBC Driver in Eclipse 87
Appendix C: Licensing 91
Appendix D: CM WebClient Support 92
Appendix E: CM WebClient Mobile 93

1. Introduction

CM WebClient is an extension to the CA Plex (Plex) development environment. It allows CA Plex-developed Java GUI applications to be deployed as web applications with minimal modifications.

In order to use CM WebClient to deploy an application, you must:

- Have an application that is configured for and operational as a Plex Java client application
- Have a Java EE-compliant web application server
- Build and deploy the Plex Java Client application using the Eclipse development environment

CM WebClient consists of two principal components:

- A template generator which runs during the generate and build process. For each Plex GUI panel, a file is generated which contains the HTML and JavaScript code needed to display the panel as a web page. The generated templates can be customized to alter the look and feel, implement new controls, and provide additional functionality to the panels.
- A servlet that is loaded into the Java EE application server at runtime. The servlet loads the Plex-generated Java application's class files and allows them to communication to and from the web user using the generated templates.

What's New in CM WebClient 1.8.2

CM WebClient 1.8.2 includes several new features.

1. Save and retrieve grid column sequence in local storage
2. Add TimeTextbox editor in grid
3. Fix shrinksafe to escape unicode
4. Fix modified event for WebEdit.ctrl on tab keypress
5. Avoid unnecessary evaluation of command arguments when \!If condition is false, prevents unrecognized command errors
6. Add z-index parameter for WebMultilineEdit.ctrl so Plex developer can specify the z-index of the control
7. Add single data type
8. Fix WebClient template renderer

9. Fix focus issue for Plex if focus statement
10. Add DateEditor in grid
11. Change setlabel to innerHTML in WebPushbutton.ctrl to fix overwriting the innertext of a span
12. Fix MobileAlerts handling
13. Add DefaultCSS to LiteralHTML.ctrl
14. Add SetState action to update button text
15. Fix not able to focus on tree label node when it is invisible
16. Fix combobox filtering
17. Allow Websydian Express to read properties file
18. Support Websydian Express v3.6
19. Add set to uppercase for password field
20. Remove tooltip from grid feature
21. Return alerts to previous panel after call return
22. Fix editable grid scroller
23. Add a space between an image and text
24. Disable combo box value set to blank when backspace key is pressed
25. Remove resizing grid content for IE
26. Clear pending updates for grid
27. Add gained focus in grid cell
28. Add longdouble type to wcli.form.getFormatValue
29. Add checking tree child node before focusing
30. Disable opening multiple combobox at the same time
31. Add DialogInit attachpoint to reposition detailpopup before it is displayed
32. Fix disable select grid when grid select event is disabled

Current Limitations

CM WebClient is currently not able to produce a web version of every Plex Java function. There are limitations related to the web environment that may require re-engineering of certain features. In order for CM WebClient to work, your application must:

- Not attempt to interact directly with the local file system or computer.
- Not use Java beans or ActiveX controls unless they have been adapted with a custom WebClient template. (See Bean controls for more information.)
- Be generated using using a supported version of CA Plex (currently 6.1 and 7.0).
- Only use panels of a supported type (currently `Dialog`, `Child Panel`, and `Top Application`).
- Note that Top Application type has a limited support. It can function as a child in a child site.

Using the Live Installer

If you do not already have an Eclipse environment, you may want to use the WebClient Live Installer to get set up. The Live Installer will install Eclipse, an instance of the Jetty application server, an instance of the Derby embedded SQL database, and a sample model and workspace. This is only recommended for test drive/demonstration purposes as some features of Eclipse like software installation are not available.

2. Installation Requirements

- Eclipse IDE for Java EE Developers 3.6 (Helios) or higher
  - Eclipse IDE
- CM WebClient 1.8.0 or higher
  - CM WebClient
- CM WebClient 1.8.0 builder plugin or higher
  - CM WebClient
- PlexBuild project
  - CM WebClient
- CM WebClient 6.0 Group Model (can be upgraded)
  - CM WebClient
- Apache Tomcat 6.0 or higher
  - Apache Tomcat
- CA Plex v6.1 or higher
- Java JDK 6 or higher is required.
  - Java JDK

3. Plex Model Configuration

Overview

In order for Plex Java functions to work with CM WebClient, each panel function must inherit from a WebClient base function. This function is called **~WebShell** and it is located in the WebClient group model. This section explains how to set up CA Plex local model to work with the WebClient pattern libraries. This document applies to CA Plex version 6.1 and above.

Instructions

The actions described in this document must be performed the first time you use the WebClient patterns in CA Plex models.

The first thing to do is to add WebClient group model to the host group model of your application. This action is performed once per group model in the same way as adding any other library models to a host model. The Web Client library may be added during creation of a group model together with the CA Plex Pattern libraries and other library models needed by the model, or they can be added to an existing CA Plex group model.

Once WebClient library is added to the group model and a local model has been defined, the objects in WebClient library models must be extracted to the local model.

All Plex functions with panel should inherit from WEBCLIENT/~WebShell before generation. The FNC is `~WebShell` should be the first triple in the pattern inheritance.

CM WebClient can only support the Dialog and Child Panel window types. For TabStrip and FrameProperty, if the function has `FNC is ACTIVE/TabStrip`, the function should inherit from `FNC is WEBCLIENT/~TabStrip` directly. If the function has `FNC is UISTYLE/FrameProperty`, the function should inherit from `FNC is WEBCLIENT/~FrameProperty` directly.

To display web messages in web page, a Plex function should inherit from WebMessages (which should inherit from ~WebUI). WebMessages is a predefined page template that can serve as an example for your own page templates that display messages.

For example: XXX is a FNC WebMessages

WebMessages is a WEBCLIENT/~WebUI

WebMessages impl name NME WebMessages

In the inheritance path for a WebClient function, the root template(WebShell) must precede page templates(WebMessages) for WebMessages to work. The Plex function should have triple inheritance path in the following order.

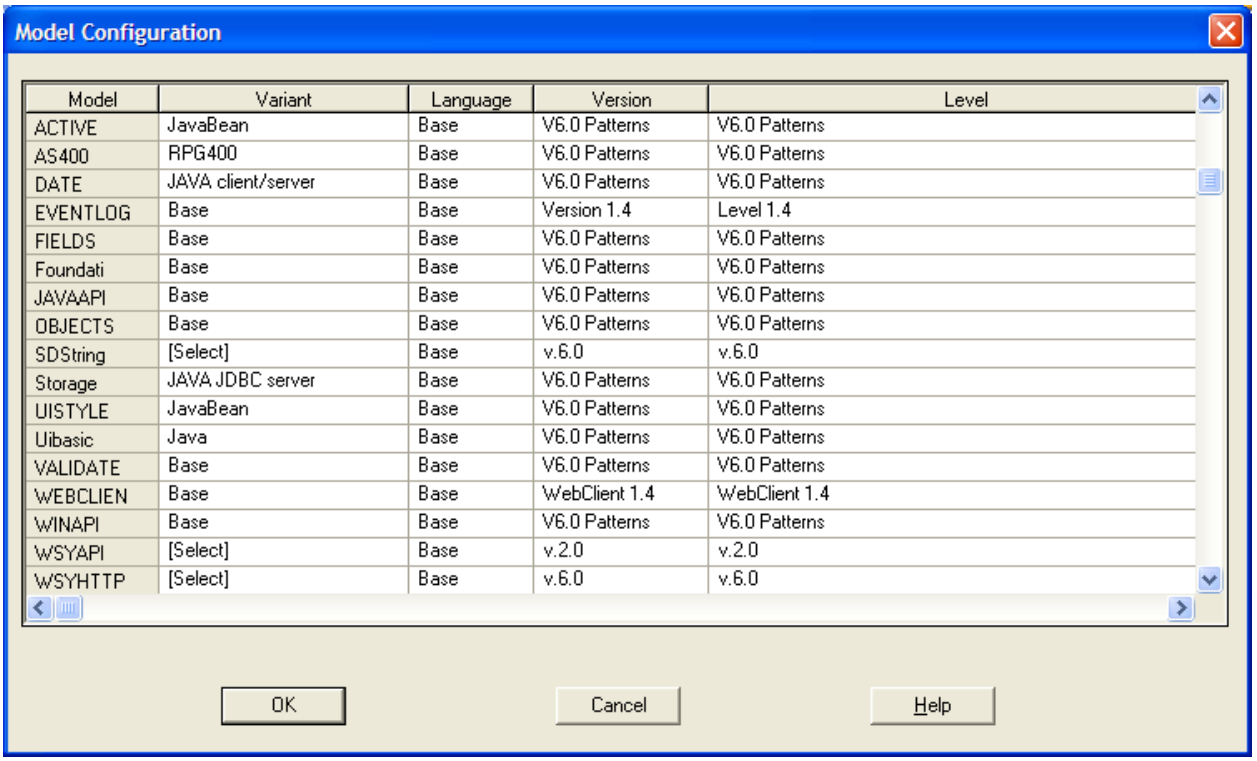
For example: XXX is a FNC WEBCLIENT/~WebShell

XXX is a FNC WebMessages

To enable exporting grid records in a csv format, a Plex function should inherit from WebExportGrid.

For example: XXX is a FNC WebExportGrid

Plex v6.1/Plex v7.0 Model Configuration



4. Overview of Workspace Setup

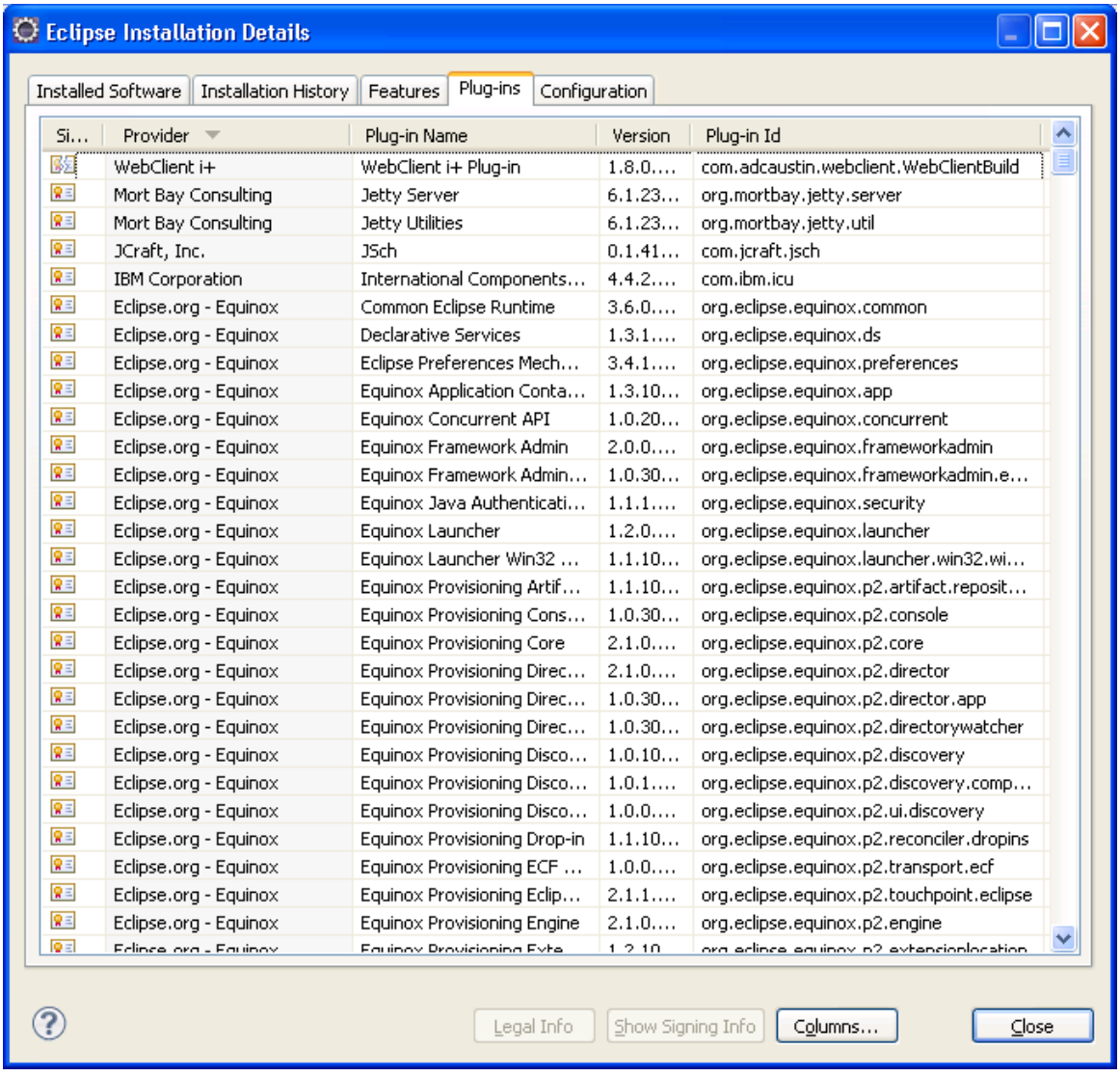
After you have the software requirements installed, below are the steps that you have to perform to setup your workspace for CM WebClient from scratch. Each step will be discussed in details when you click on the link.

1. Install CM WebClient builder plugin in eclipse plugin.
2. Create a web server in the workspace.
3. Create a new java project.
4. Import PlexBuild project.
5. Setup PlexBuild Run Configuration
6. Configure build.properties in your PlexBuild project.
7. Generate your Plex functions. Do not build.
8. Run PlexBuild project.
9. Import WebClient project.
10. Configure the java project.
11. Configure obclient.properties.
12. Generate web template.
13. Create a dynamic web project.
14. Configure dynamic web project.
15. Create a new servlet in web project.
16. Create a New Listener in WebClient Servelt
17. Add Plex images in the web project.
18. Run CM WebClient in Eclipse.
19. Configure WebClient.properties.

Install WebClient Builder Plugin

Copy com.adcaustin.webclient. WebClientPlugin-1.8.x.jar to eclipse\plugins folder.

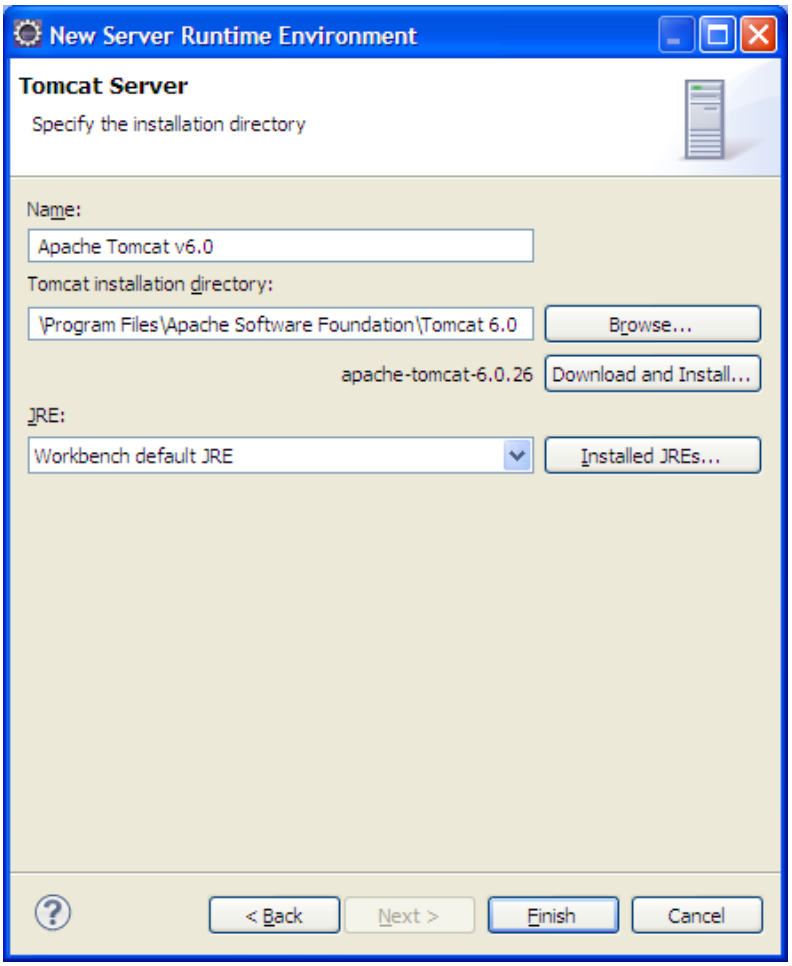
To confirm that WebClient Plugin builder is installed correctly.
Go to **Help** → **About Eclipse** → Click on **Installation Details** → Sort **Plug-in Id** column in ascending order


Eclipse Installation Details
Installed Software
Installation History
Features
Plug-ins
Configuration
Si...
Provider
Plug-in Name
Version
Plug-in Id
WebClient i+ | WebClient i+ Plug-in | 1.8.0.... | com.adcaustin.webclient.WebClientBuild
Mort Bay Consulting | Jetty Server | 6.1.23... | org.mortbay.jetty.server
Mort Bay Consulting | Jetty Utilities | 6.1.23... | org.mortbay.jetty.util
JCraft, Inc. | JSch | 0.1.41... | com.jcraft.jsch
IBM Corporation | International Components... | 4.4.2.... | com.ibm.icu
Eclipse.org - Equinox | Common Eclipse Runtime | 3.6.0.... | org.eclipse.equinox.common
Eclipse.org - Equinox | Declarative Services | 1.3.1.... | org.eclipse.equinox.ds
Eclipse.org - Equinox | Eclipse Preferences Mech... | 3.4.1.... | org.eclipse.equinox.preferences
Eclipse.org - Equinox | Equinox Application Conta... | 1.3.10... | org.eclipse.equinox.app
Eclipse.org - Equinox | Equinox Concurrent API | 1.0.20... | org.eclipse.equinox.concurrent
Eclipse.org - Equinox | Equinox Framework Admin | 2.0.0.... | org.eclipse.equinox.frameworkadmin
Eclipse.org - Equinox | Equinox Framework Admin... | 1.0.30... | org.eclipse.equinox.frameworkadmin.e...
Eclipse.org - Equinox | Equinox Java Authenticati... | 1.1.1.... | org.eclipse.equinox.security
Eclipse.org - Equinox | Equinox Launcher | 1.2.0.... | org.eclipse.equinox.launcher
Eclipse.org - Equinox | Equinox Launcher Win32 ... | 1.1.10... | org.eclipse.equinox.launcher.win32.wi...
Eclipse.org - Equinox | Equinox Provisioning Artif... | 1.1.10... | org.eclipse.equinox.p2.artifact.reposit...
Eclipse.org - Equinox | Equinox Provisioning Cons... | 1.0.30... | org.eclipse.equinox.p2.console
Eclipse.org - Equinox | Equinox Provisioning Core | 2.1.0.... | org.eclipse.equinox.p2.core
Eclipse.org - Equinox | Equinox Provisioning Direc... | 2.1.0.... | org.eclipse.equinox.p2.director
Eclipse.org - Equinox | Equinox Provisioning Direc... | 1.0.30... | org.eclipse.equinox.p2.director.app
Eclipse.org - Equinox | Equinox Provisioning Direc... | 1.0.30... | org.eclipse.equinox.p2.directorywatcher
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.10... | org.eclipse.equinox.p2.discovery
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.1.... | org.eclipse.equinox.p2.discovery.comp...
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.0.... | org.eclipse.equinox.p2.ui.discovery
Eclipse.org - Equinox | Equinox Provisioning Drop-in | 1.1.10... | org.eclipse.equinox.p2.reconciler.dropins
Eclipse.org - Equinox | Equinox Provisioning ECF ... | 1.0.0.... | org.eclipse.equinox.p2.transport.ecf
Eclipse.org - Equinox | Equinox Provisioning Eclip... | 2.1.1.... | org.eclipse.equinox.p2.touchpoint.eclipse
Eclipse.org - Equinox | Equinox Provisioning Engine | 2.1.0.... | org.eclipse.equinox.p2.engine
Legal Info
Show Signing Info
Columns...
Close

Configure Apache Tomcat in Eclipse

a. Create a Tomcat installed runtime.

Window → **Preferences** →**Server** → **Installed Runtimes** → Click on **Add…** → Select **Apache Tomcat v6.0** under Apache → Click **Next** > → Browse to your Apache Tomcat installation directory.



Note: Ensure the selected **JRE** is a full JDK and is of a version that will satisfy Apache Tomcat (this scenario was written using SUN JDK 1.6.0_02). If necessary, you can click on **Installed JREs …** to add JDKs to Eclipse. Click **Finish**.

b. Start Tomcat server.
Window → **Show View** → **Servers** → Right-click on the server window →Select **New** → **Server** → Select **Apache** → **Tomcat v6.0 server** → Click **Finish** → Right-click on the server → Click **Start**.

Create a New Java Project

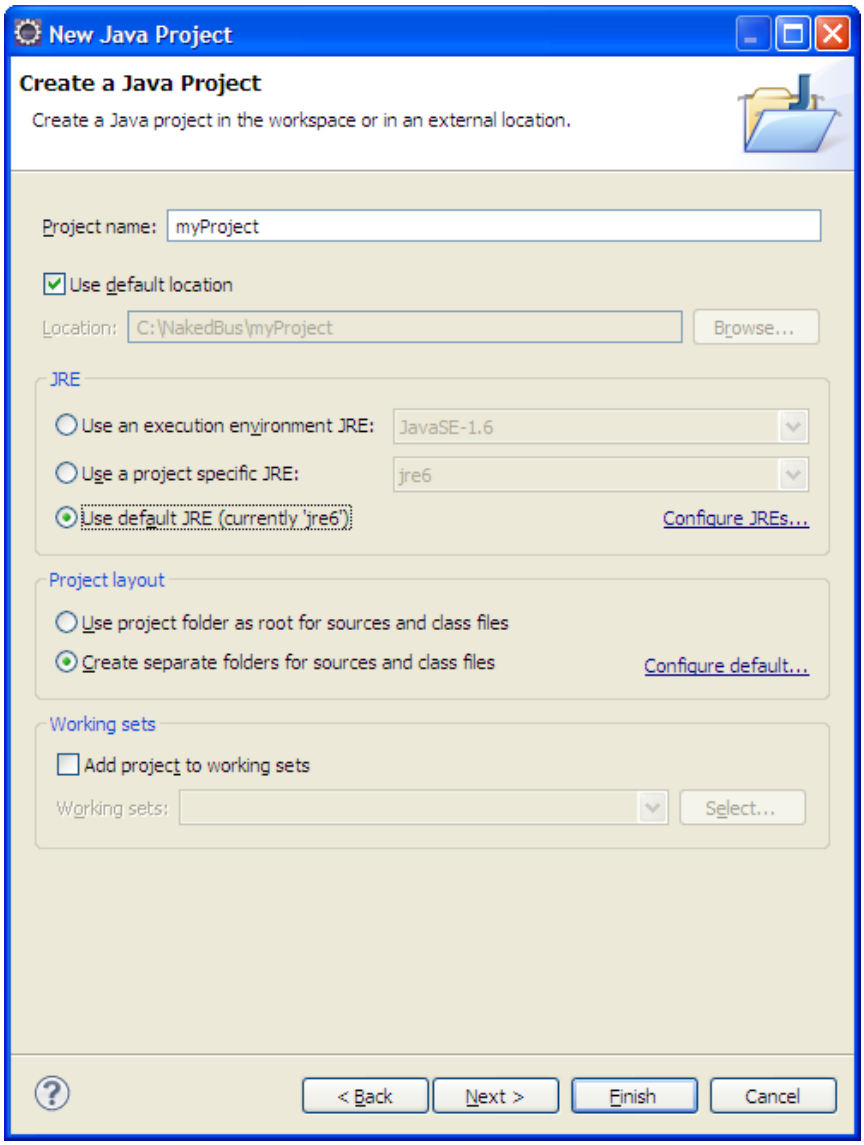
Introduction

A Java project contains the generated Plex Java source code, resource files and generated web templates.

Instructions

a. **File** → **New** → **Other…** → Expand **Java** folder → Select **Java Project** → Click **Next >**

b. Enter your project name in the project name field → Select **Use default JRE (currently 'jre6')** → click **Finish**.



c. Create a new resource folder named PlexResource.
Right-click on your java project → Select **New** → Select **Source Folder** → Enter *PlexResource* as the folder name → Click **Finish**.

Import PlexBuild project

Introduction

PlexBuild is an Eclipse project that contains an Apache Ant script to move Plex Java source code into Eclipse for compilation. Ant allows Eclipse to create and run Ant buildfiles from the Eclipse workbench. On the classpath tab of the Ant Runtime preference page, additional classes defining tasks and types can be added to the Ant classpath.

Instructions

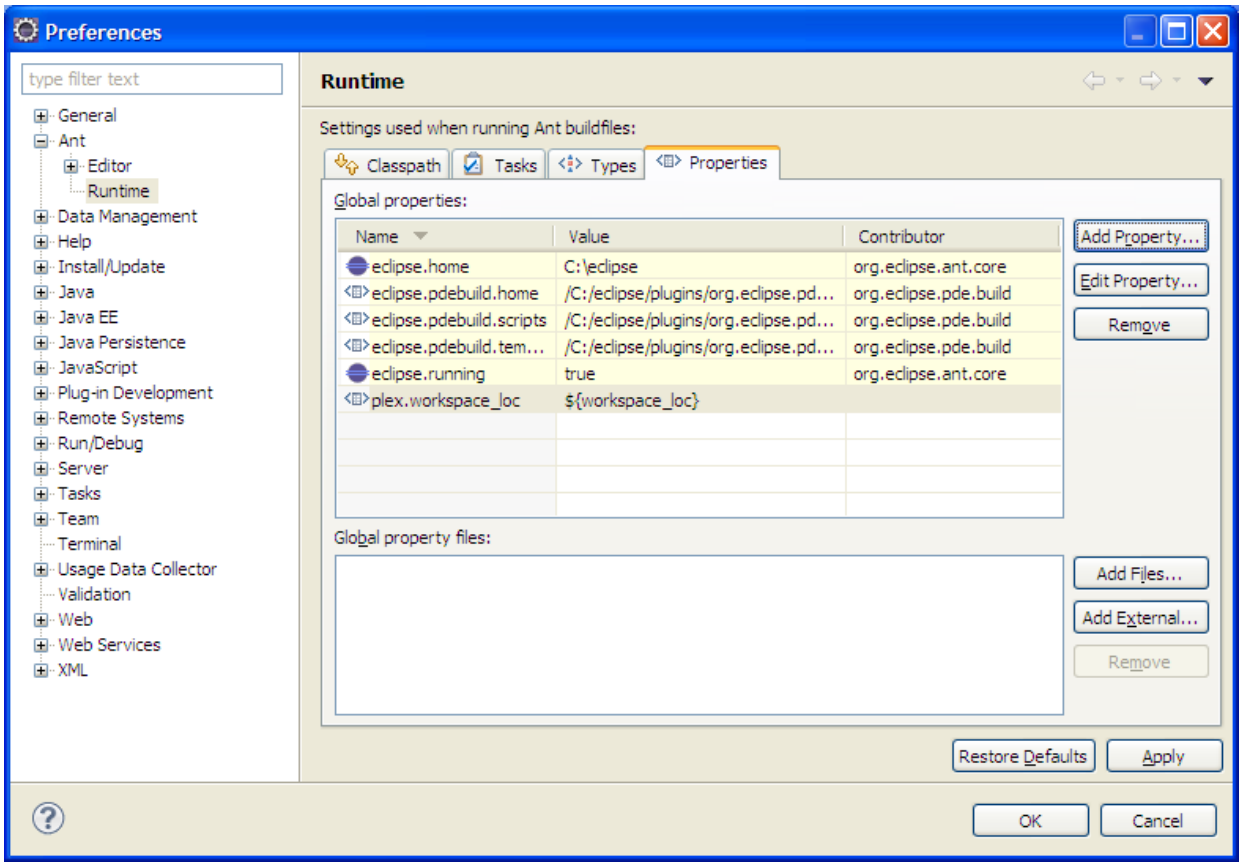
a. **File** → **Import**… → Expand **General** folder → Select **Existing Projects into Workspace** → Click **Next >**

b. If you have the project in a file directory, use **Select root directory**. If you have the project in a zip file format, use **Select archive file**. Browse to your directory and click **Finish**.

Setup PlexBuild Run Configuration

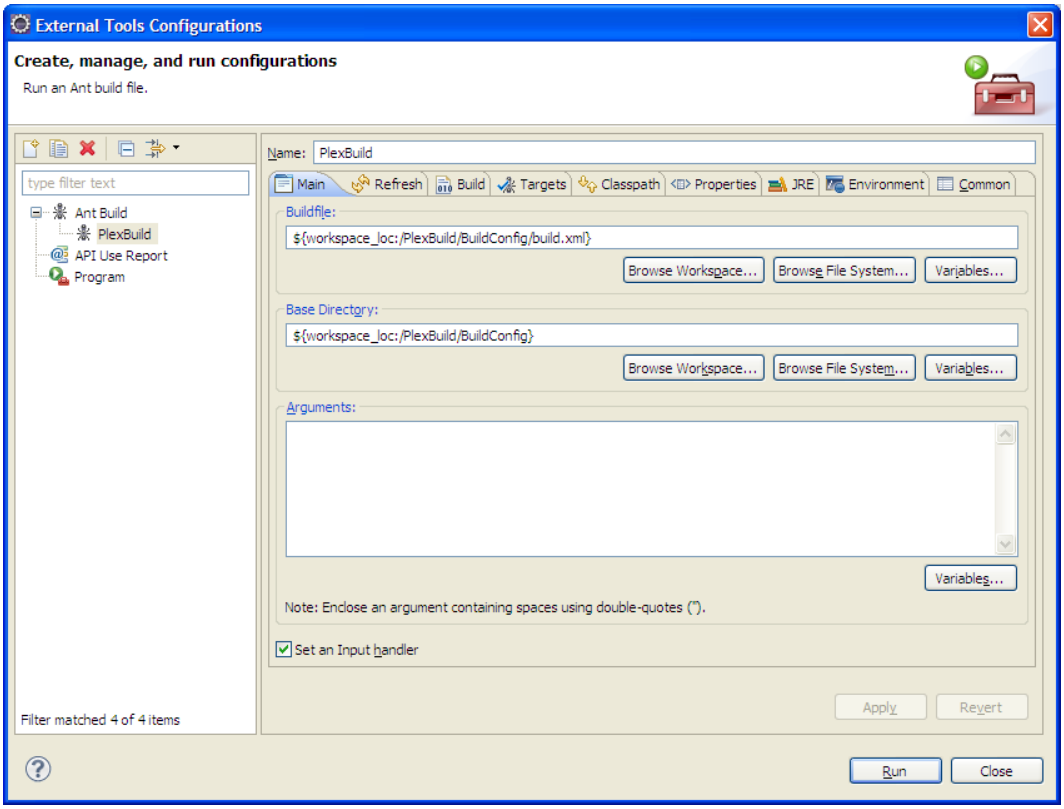
a. Modify Ant classpath globally
Window → **Preferences** → **Expand Ant** → Select **Runtime** → Open the **Properties** tab → Click **Add Property…**

b. Enter *plex.workspace_loc* in the Name field and *${workspace_loc}* in the Value field → Click **OK**.

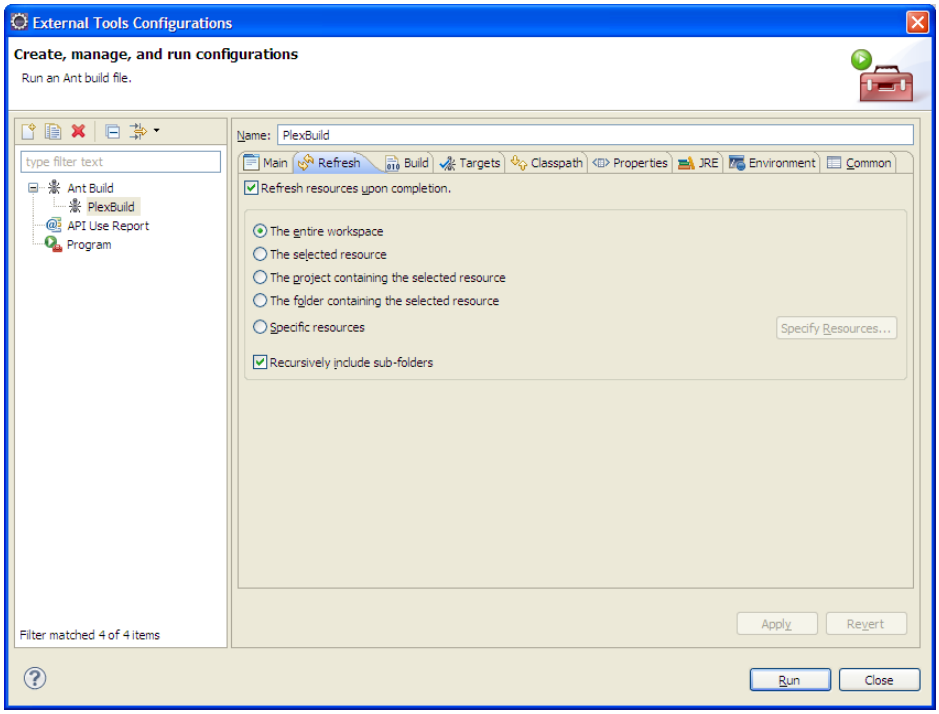


c. Create an ANT classpath for launch configuration.

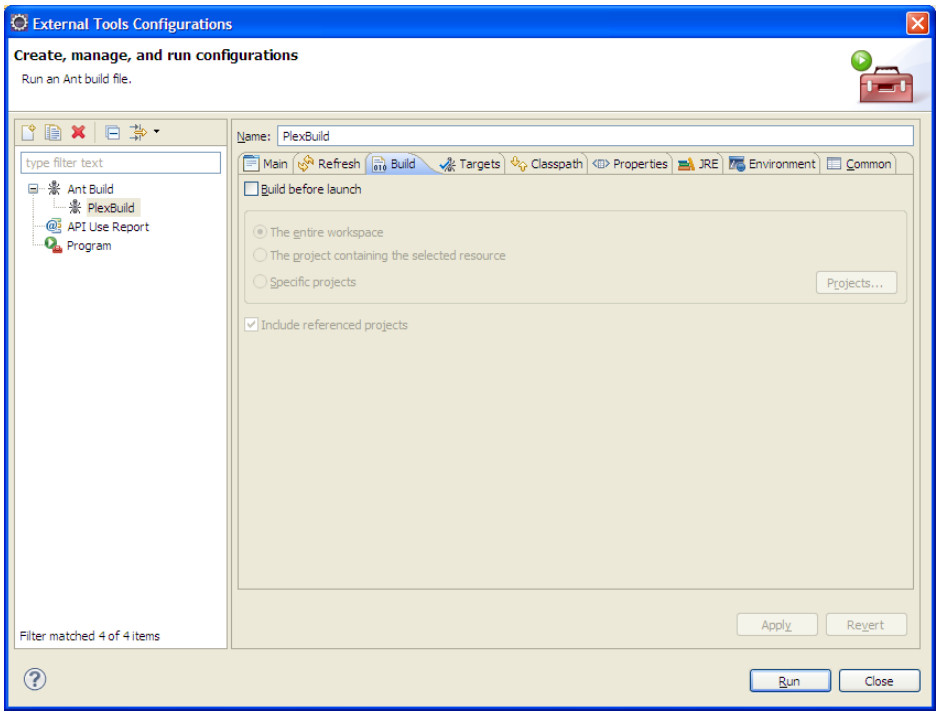
- **Run → External Tools → External Tools Configuration…** → Select **Ant Build** → Press the new button.
- Name the new configuration PlexBuild. Set the Buildfile to *${workspace_loc:/PlexBuild/BuildConfig/build.xml}* and Base Directory to *${workspace_loc:/PlexBuild/BuildConfig}*
- Click **Apply** to save.



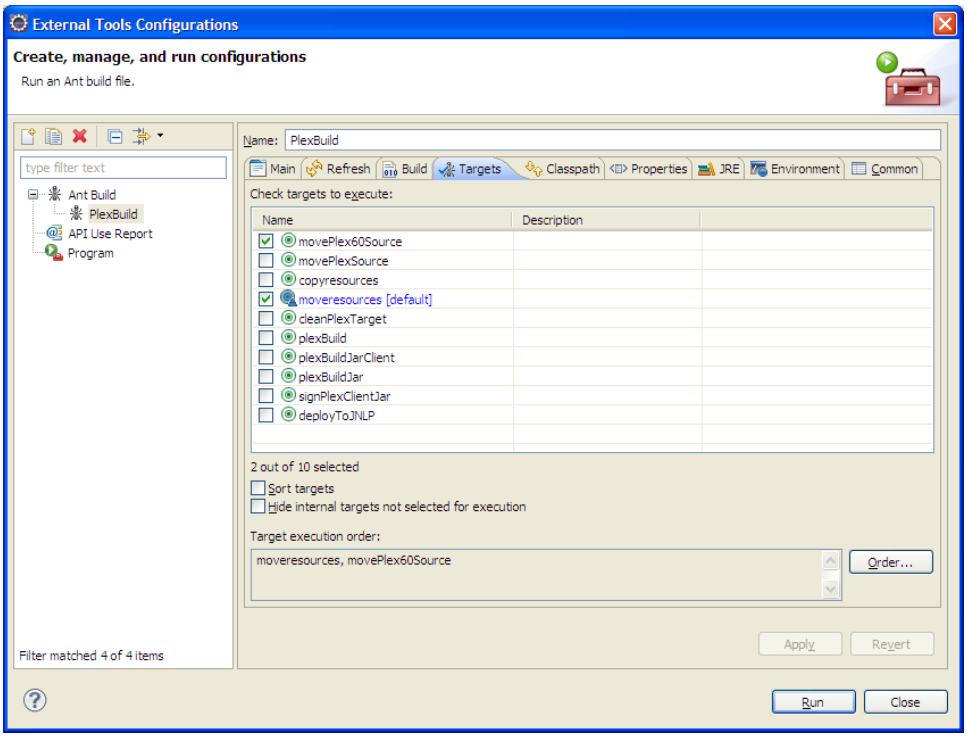
- Open the **Refresh** tab → Check on **Refresh resources upon completion**.
- Click **Apply** to save.



- Open the **Build** tab → Uncheck **Build before launch.**
- Click **Apply** to save.



- Open **Targets** tab → Check move *Plex60Source* and *moveresources[default]* as targets to execute.
- Click **Apply** to save.


External Tools Configurations
Create, manage, and run configurations
Run an Ant build file.
type filter text
Ant Build
PlexBuild
API Use Report
Program
Filter matched 4 of 4 items
Name: PlexBuild
Main
Refresh
Build
Targets
Classpath
Properties
JRE
Environment
Common
Check targets to execute:
Name
Description
movePlex60Source
movePlexSource
copyresources
moveresources [default]
cleanPlexTarget
plexBuild
plexBuildJarClient
plexBuildJar
signPlexClientJar
deployToJNLP
2 out of 10 selected
Sort targets
Hide internal targets not selected for execution
Target execution order:
moveresources, movePlex60Source
Order...
Apply
Revert
Run
Close

Configure build.properties File in PlexBuild project

a. Expand PlexBuild project → Expand **BuildConfig** folder → Right-click on build.properties file and select open with Text Editor.

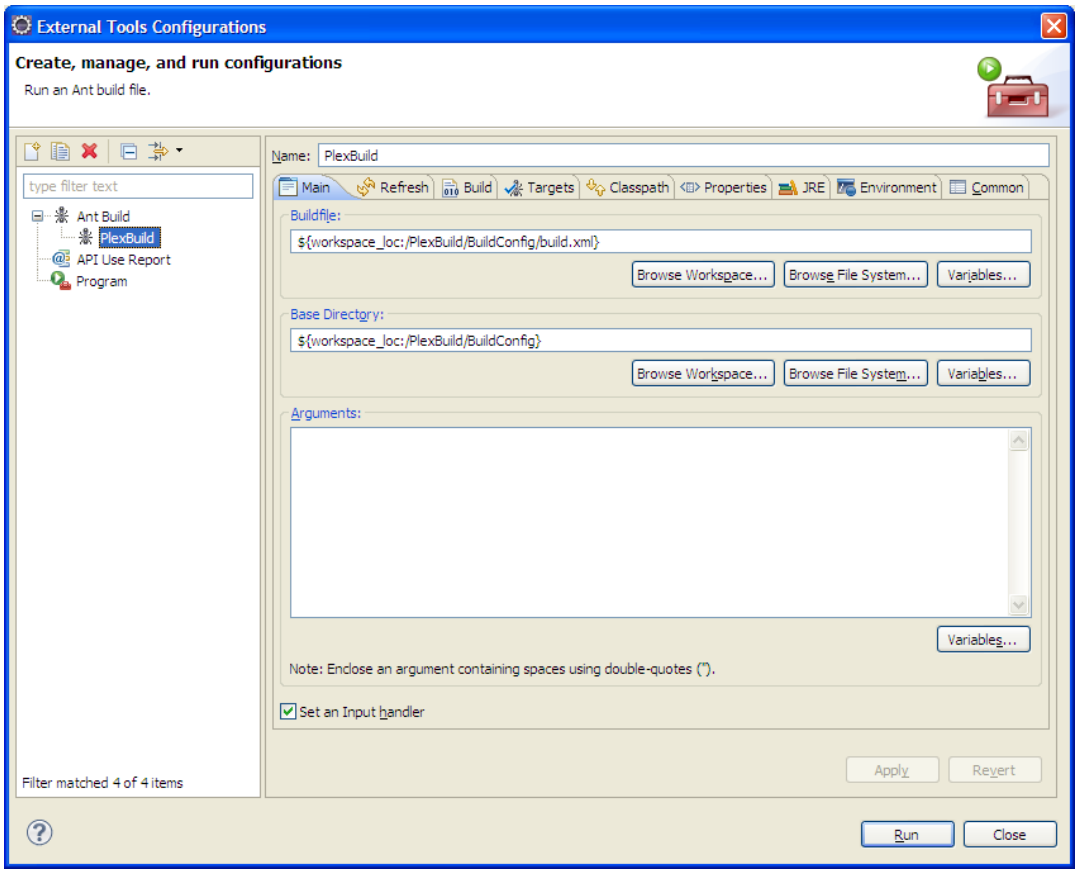
b. Find the line *plex.import.project_name=XXX*. Edit the line so that *plex.import.project_name=*<your java project name>

c. Find the line *plex.gen.root=XXX*. Edit the line so that *plex.gen.root=*<your gen directory>
Note that forward slashes will be used instead of backward slashes.

d. Save the file.

How to Run PlexBuild in Eclipse

a. Go to **Run** → **External Tools** → **External Tools Configuration…**

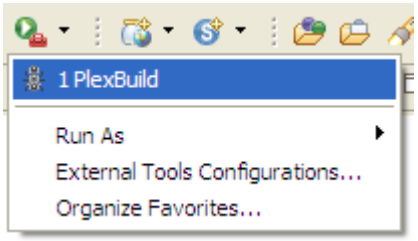


b. Click **Run**.

You should see on your console that the functions are successfully built and moved to your project source folder.

```
Buildfile: C:\Webplex\PlexBuild\BuildConfig\build.xml
movePlex60Source:
     [move] Moving 261 files to C:\Webplex\Plex60Project\src
     [move] Moving 46 files to C:\Webplex\Plex60Project\src
BUILD SUCCESSFUL
Total time: 6 seconds
```

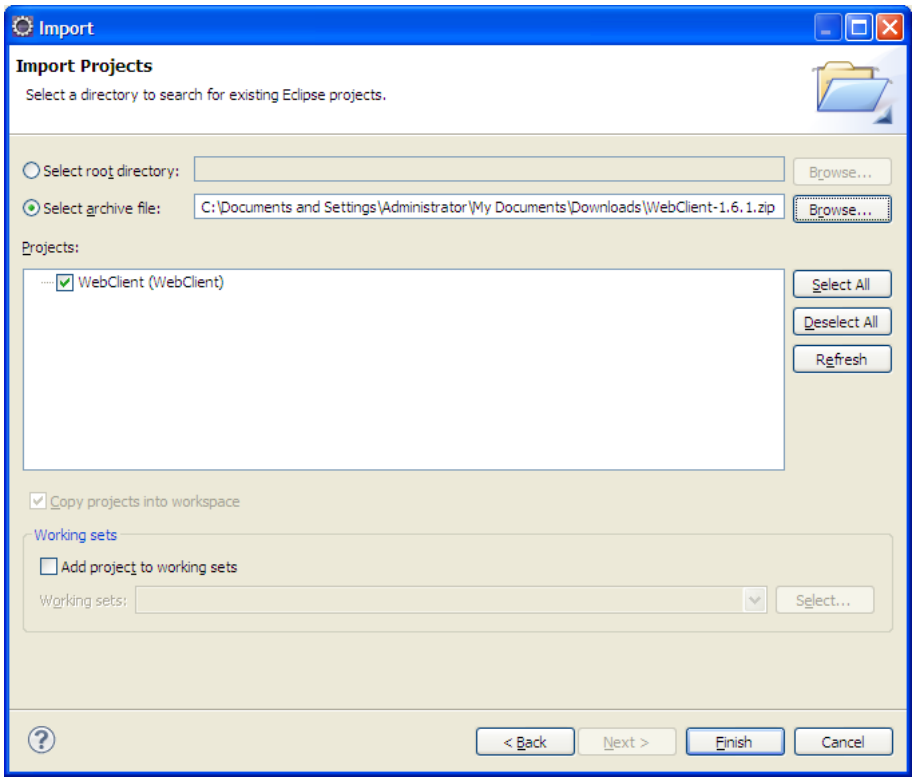
c. The next time you run PlexBuild, you can click on icon shown below.



Import WebClient project

a. **File** → **Import**… → Expand **General** folder → Select **Existing Projects into Workspace** → Click **Next** >

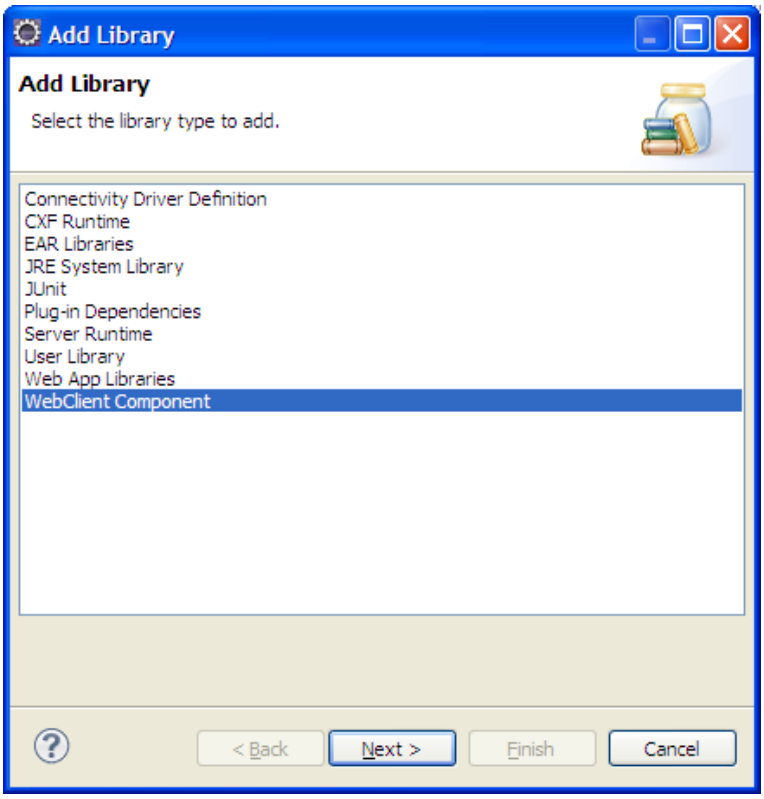
b. If you have the project in a file directory, use *Select root directory*. If you have the project in a zip file format, use *Select archive file*. Browse to your directory and click **Finish**.



Configure Java Project

a. Right-click on your java project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add Library…**

b. Select **WebClient Component** → Click **Next** > → Check **WebClient** → Click **Finish**.



c. Import obclient.properties to your java project souce folder <java project>\src folder.
File→ **Import**→ Expand **General** folder → Select **File System** → Click **Next >** → Browse to ObJava directory under CA Plex folder and select obclient.properties → Click **Finish**.

If you have tab functions, you also need to add pvAll.jar in your java project build path.

d. Right-click on your java project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add External JARs…** → Browse to CA Plex browse to C:\Documents and Settings\All Users\Documents\CA\Plex**ver**\class**ver**\PatternLibs\ACTIVE\Beans for Plex **ver**

e. Click **Finish**.

Configure obclient.properties

Note: CM WebClient date and time format works based on browser locale instead of obclient.properties date and time format.

a. Open obclient.properties with Text Editor.
Right-click on obclient.properties → Select **Open With** → Select **Text Editor**.

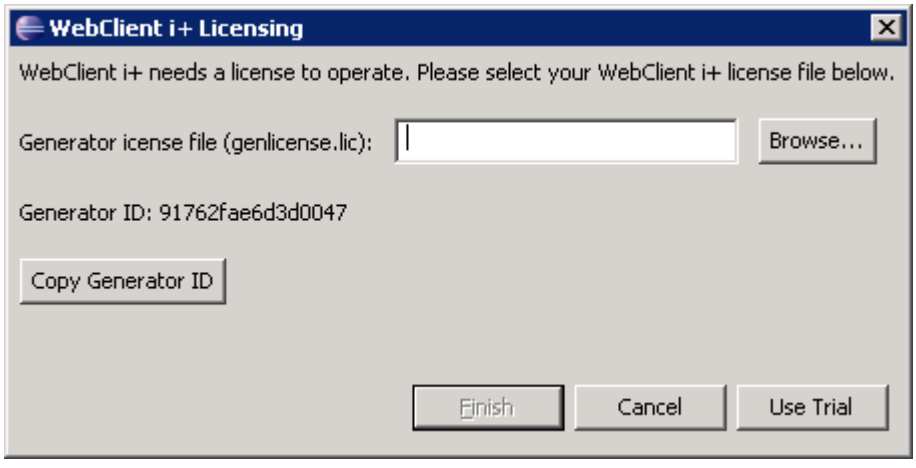
The file is configured differently depending on what driver and data source you are using. It determines which database and port to connect to. The example below is using SUN JDBC driver.

```
Environment.Default.Driver=sun.jdbc.odbc.JdbcOdbcDriver
Environment.Default.DataSource=jdbc:odbc:WebClient
Environment.Default.User=username
Environment.Default.Password=password
```

b. Save the changes.

Generate WebClient Templates

a. Right-click on your java project → Select **WebClient** → **Enable Template Builder**.
WebGenTemplates folder will be automatically created under your java project. You will be prompted to enter your genlicense.lic. Click on Use Trial or enter your license file and click Finish.



The console should display "Generating Web templates" with no error message.

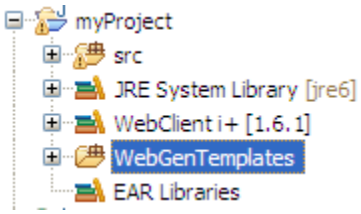
```
>>> Starting WebClient Build for wxpcourse60

> Using WebClient i+ v1.8
> Licensed to Trial User
> Valid until Sun Apr 15 00:00:00 BST 2012

Generating Web templates.
..................................................
.................

>>> WebClient Build complete.
    38 panels processed (with 1 diagnostics.) 0 panels not processed.
```

b. You should have WebGenTemplates folder created under your java project.



Note: If there is no web template generated, check if there is a PlexResource folder created. If there is, make sure PlexResource is a source folder and there are .panelresource and .resource files associated to the function name.

Create a New Dynamic Web Project

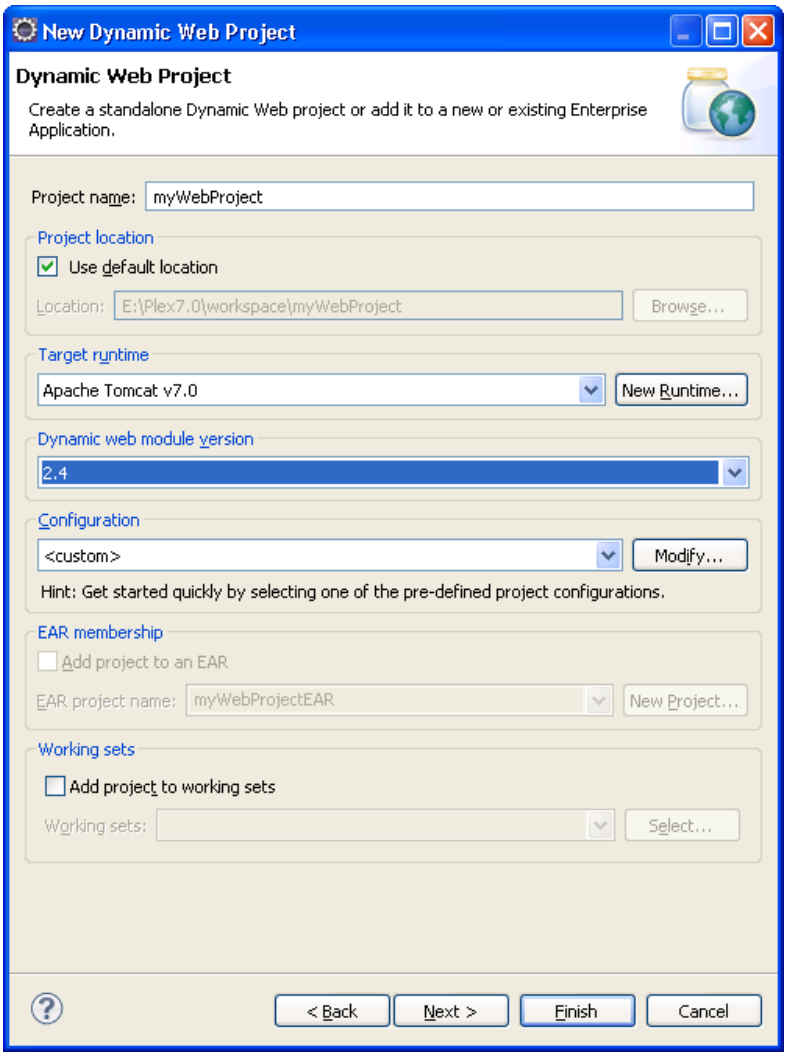
Introduction

A Dynamic Web Project contains JSP (Java Server Pages) files, HTML files, servlets, and Plex runtime JAR files. The web project is then deployed on Apache Tomcat to be viewed on a browser. To avoid any unexpected issue, we recommend that a new server is created in the workspace before creating a new web project.

Instructions

a. **File** → **New** → **Other…** → Expand **Web** folder → Select **Dynamic Web Project** → click **Next** >

b. Enter your web project name in the project name field → Select target runtime server → Dynamic web module version should be set to **2.4** → Click **Finish**.

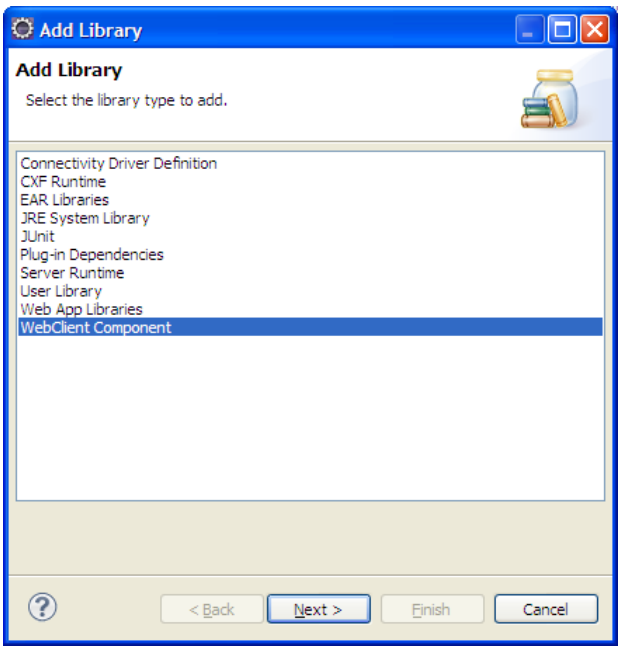


Expand your web project/WebContent/WEB-INF folder to make sure that there is web.xml file created.

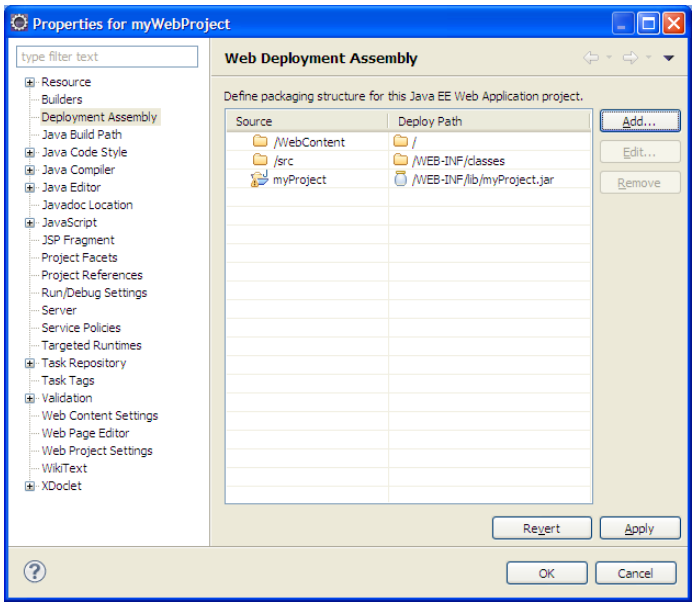
Configure Build Path for the New Web Project

a. Right-click on your web project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add Library…**

b. Select **WebClient Component** → Click **Next** > → Check **WebClient** → Click **Finish**.



c. Select **Deployment Assembly** on Properties window → Click **Add…** → Select **Project** → Click **Next** > → Select your java project → Click **Finish**. If you have any additional jar files that are required for deployment, you need to add them in **Deployment Assembly**.



Create a New Servlet in Web Project

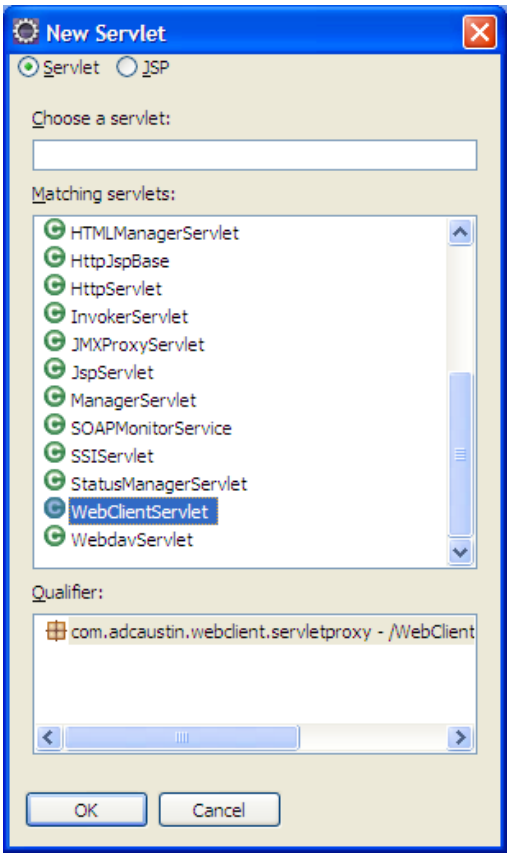
Introduction

WebClientServlet is a method of deploying the CM WebClient architecture. It consists of a servlet, called WebClientServlet, which launches your WebClient Plex application within a servlet container, and a configuration file, called WebClient.properties, which contains settings for the servlet.

In this mode, a single Plex function is the top-level component of your web application, known as the entry point. Consequentially, WebClientServlet is not suitable when you wish to launch a Plex function as a Business Process in Websydian Express. If you are using Websydian Express, see chapters Deploying CM WebClient with Websydian Express for Java.

Instructions

a. Right-click on your web project → Select **New** → **Servlet** → Check "Use an existing Servlet class or JSP" → Select **WebClientServlet** → Click **OK**.



b. Click **Finish** in Create Servlet window.

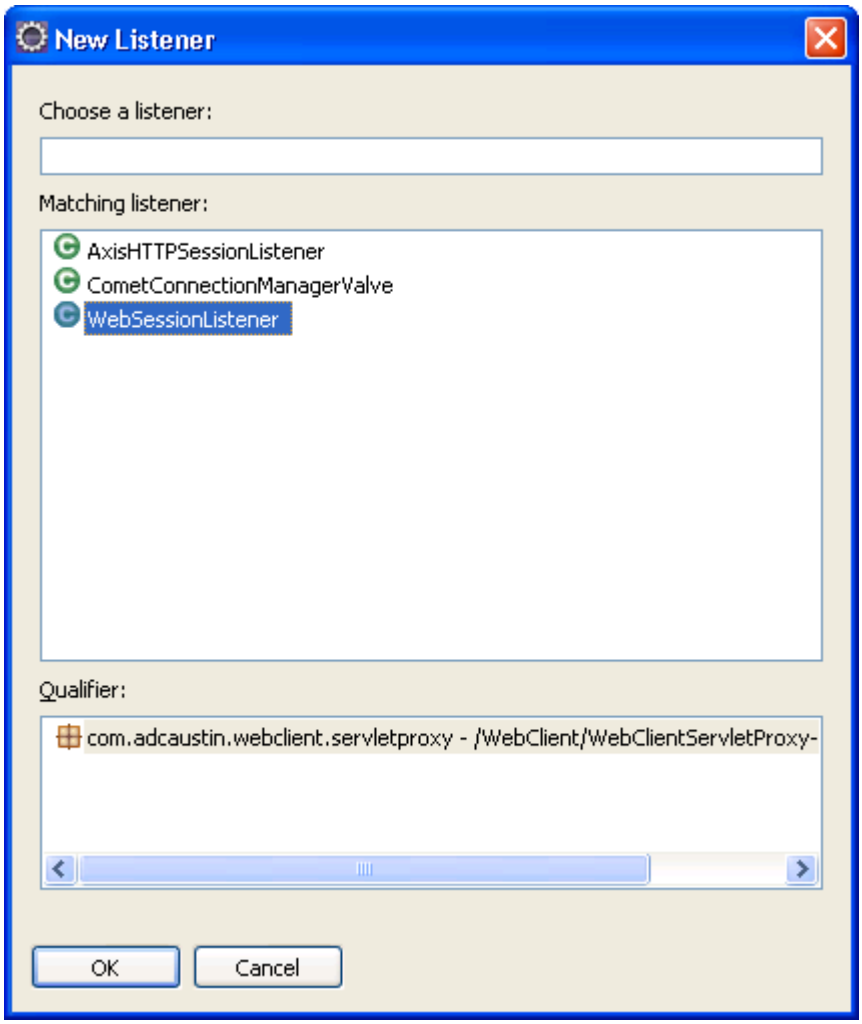
Note: If WebClientServlet is not showing up on your servlet list, add WebClient project in the Deployment Assembly under your web project.

Right-click on your web project → select **Properties** → select **Deployment Assembly** → Click **Add…** button → select **Project** → select WebClient project → click **Finish**.

Create a New Listener in WebClient Servlet

Instructions

a. Right-click on your web project → Select **New** → **Listener** → Check "Use existing listener class" → Select **WebSessionListener** → Click **OK**.



b. Click **Finish** in Create Listener window.

Add Plex Images to the Web Project

a. Create an images folder under WebProject/WebContent folder.

b. Copy/Import all images files in the images folder.

Note: Keep in mind that CM WebClient is case-sensitive. Make sure that your Plex panel properties and your image filename is a match.

Run WebClient in Eclipse

a. Right-click on your tomcat server in the server tab → Select **Publish**.

b. After it is finished publishing → select **Start**.

c. Open your browser and go to http://localhost:8080/<contextroot>/<servletname>

To find your context root, right-click on your web project → select **Properties** → Select **Web Project Settings**.

To find your servlet name, expand your web project → expand WebContent folder → expand WEB-INF folder → open your web.xml with Text Editor

When you first deploy the application, you should see a page that let you download WebClient.properties. If it gives an error that says "Unable to validate WebClient license", use this link http://127.0.0.1:8080/<contextroot>/<servletname>

Welcome to WebClient i+
http://localhost:8080/myWebProject/WebClientServlet
Welcome to WebClient i+
Congratulations! You have succesfully installed WebClient i+. WebClient uses a new property file format now, called WebClient.properties. Based on your server configuration, we have automatically generated a default version of the configuration files below:
WebClient.properties
To finish configuring your application, please download this file and place it in the src folder of your Web Project (or another project that is on your application's classpath).Edit the file by specifying a value for the webclient.entry property and (optionally) set up a log file. You may then publish and restart your application.
Reload page
WebClient i+ v1.6.1
Copyright 2010 Advanced Development Centers of America, In

Click on WebClient.properties and save the file in your java or web project source file.

Configure WebClient.properties

a. In the file *WebClient.properties*, located in the src folder of either your Java Project or your Dynamic Web Project, change the properties parameters as follows.

licensingserver.ipaddr
The ipaddress of the machine where the licensing server is started. Only uncomment this property if you have WebClient licensing server installed.
For example:
```
licensingserver.ipaddr=localhost
```

licensingserver.tcpport
The TCP port on which licensing server listens. Only uncomment this property if you have WebClient licensing server installed. The default port is 65000.
For example:
```
licensingserver.tcpport=65000
```

licensingserver.licensefile
If there is no valid license file, this property must be commented. CM WebClient will automatically activate 10-user developer license. If there is a license file, then specifies the complete file system path to the license file(license.lic)
For example:
```
licensingserver.licensefile=C\:/Program Files\WebClient/license.lic
```

log4j.logger.com.adcaustin.WebClient
The granularity of the log messages. Legal values are: DEBUG, INFO, WARN, ERROR, FATAL. DEBUG should only be used when debugging and in test environments. In a production environment use WARN or ERROR. For example:
```
log4j.logger.com.adcaustin.WebClient=INFO, WebClientServlet
```

log4j.appender. WebClientServlet.file
specifies the path and the name of the log file. For example:
```
log4j.appender.WebClientServlet.file=C:/WebClient.log
```

webclient.entry : <packagename>.<Plex function implementation name>
The entry point to the Web Application; i.e., the initial function to be called when a new session is created. Specify the full name of the class including package names. For example:
`webclient.entry=WXPCOURSE60.WXuF`

webclient.entry.timeout
The entry point that is called if a session has timed out on the server and the user attempts to resume the session. For example:
`webclient.entry.timeout=WXPCOURSE60.sesstimeout`

webclient.entry.url.<path>
The entry point that is associated with a specific url. The url is determined by taking the components after the word "url", and changing dots to forward slashes (/). For example:
webclient.entry.url.pub.item=com.foo.bar causes the relative url /pub/item to launch the function com.foo.bar.
Wildcards are allowed and are specified with an asterisk(*). For example:
webclient.entry.url.api.*=com.foo.api.* causes the relative url /api/myfunc to launch the function com.foo.api.myfunc.
In both cases, the url specified is relative to the url-pattern in the servlet-mapping of the servlet in the web.xml file. The url-pattern must end with "/*" in order for this to work. For example, if the url-pattern is /webclient/*, then the url mappings above will be relative to /webclient
(e.g., /webclient/pub/item and /webclient/api/myfunc)

servlet.errorpage.url
specifies a relative URL to an error page within the context of the WAR file where the servlet is deployed. On errors the servlet will then call this error page which then is responsible of generating a response to the user. The error page have to be placed within the scope of the
WAR-file where / (slash) indicates the root of the WAR-file. For example:
`servlet.errorpage.url=/myWebProject/error.jsp`

resurl.plex.imageresources
URL reference for where the image resources as referenced by Plex are kept. This URL should contain a folder named images. For example:
`resurl.plex.imageresources=/myWebProject`

Define.WSACTION
/<DynamicWebProject>/<ServletName>. URL reference for servlet.
For example:

```
Define.WSACTION=/myWebProject/WebClientServlet
```

Define.WCTHEME
Set WebClient theme to either claro or tundra. This property is optional. If it is not set, the default value is set to tundra.
For example:

```
Define.WCTHEME=claro
```

servlet.statistics.level
Set this option to cause the servlet to gather some runtime statistics on called functions. This feature only works when calling AS/400 functions. The valid levels are as follows: 1 = Collect data on called functions grouped by function type and function name. The number of times each function is called as well as its inclusive run time. After each request, a report is logged to the servlet log at the INFO log level. For example:

```
servlet.statistics.level=1
```

b. Save changes.

c. Republish and restart your web server. Reload the page. You may need to close the browser and relaunch the app

5. Deploying CM WebClient with Websydian Express for Java

Introduction

The Websydian Express Proxy for Java (Wxp4J) is an implementation of Websydian Express where all business logic, including Websydian Express system functions, business processes, and CM WebClient functions, run inside of a Java EE-compliant servlet container. In this mode, all Plex functions are generated for the Java platform, and deployed to a web application archive (WAR) file.

Instructions

1. Complete Steps 3 – 11 of Section 4
2. Import The express30.war File
3. Configure J2eeProxy.propperties
4. Add WebClient.properties to express30
5. Configure WebClient.properties
6. Configure obusr.properties.
7. Configure express30 project.
8. Configure Websydian Express Admin site.
9. Run WebClient application.

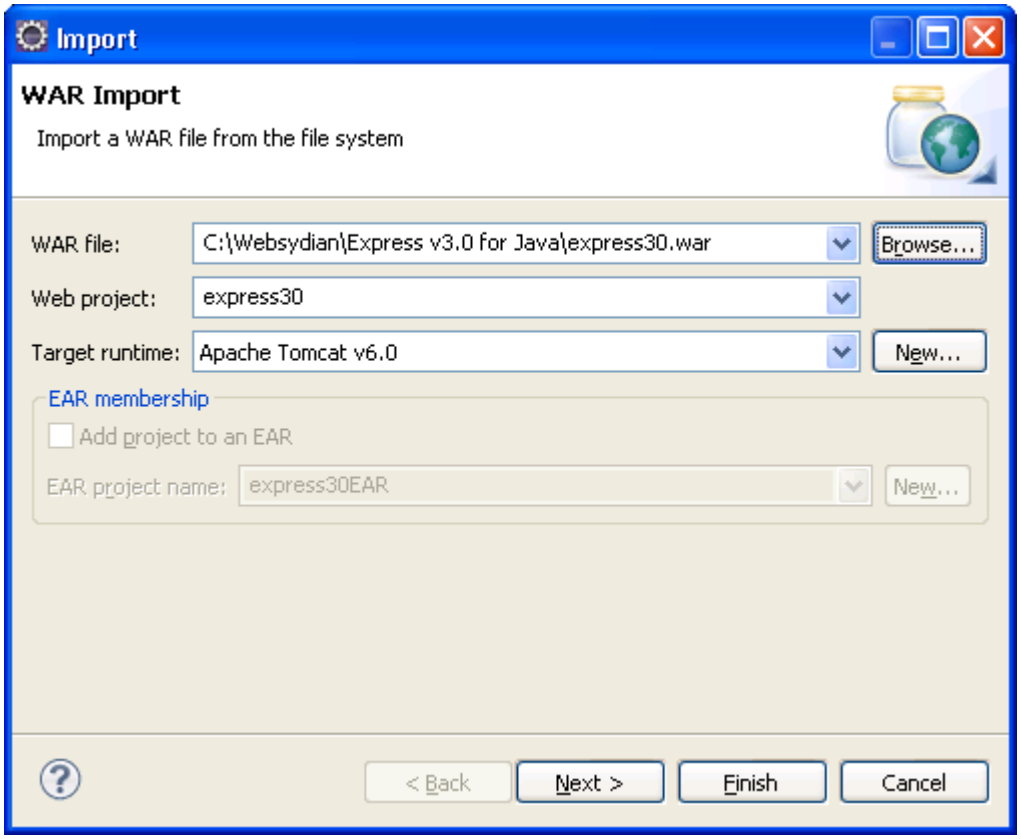
Complete Steps 3 – 11 of Section 4

3. Create a new Java project.
4. Import PlexBuild project.
5. Setup PlexBuild run configuration.
6. Configure build.properties file in PlexBuild project.
7. Run PlexBuild in Eclipse.
8. Import WebClient project.
9. Add WebClientDWA.jar and WxpServletProxy.jar to your WebClient build path/folder.
9. Configure java project.
10. Configure obclient.properties.
11. Generate web template.

Import the Express30.war File

a. **File** → **Import** → Expand **Web** folder → Select **WAR file** → Click **Next** >

b. Browse to express30.war directory. Click **Finish**.



Configure J2eeProxy.properties

The file is saved in express30/WebContent/WEB-INF.
websydianserver.log.file=, servlet.log.file=, and websydianserver.licensefilepath= should be configured based on your settings.

```
# ------------------------------------------------------------------------
---------------------------
websydianserver.log.level=ERROR
```

```
websydianserver.log.file=C:/Websydian/Express v3.0 for
Java/logs/websydianserver.log

servlet.log.level=ERROR
servlet.log.file=C:/Websydian/Express v3.0 for Java/logs/J2eeproxy.log

websydianserver.licensefilepath=C:/Websydian/Express v3.0 for
Java/license/license.key

############################################################
```

You can find a sample J2eeProxy.properties file at the Download section of: CMFirst Software and Documentation

Add WebClient.properties to Express30 Project

Copy WebClient.properties to express30/src folder.

Configure WebClient.properties.

log4j.appender.WebClientServlet.file should be configured based on your settings.

You can find a sample of a websydian WebClient.properties file at: CMFirst Software and Documentation

Configure obusr.properties.

Make sure you configure the following properties according to your settings.

```
# Current connection information
#-------------------------------------------------------------------
------------------

Environment.Default.Driver=net.sourceforge.jtds.jdbc.Driver
Environment.Default.DataSource=jdbc:jtds:sqlserver://localhost:1433/Web
sydianExpress30
Environment.Default.User=myusername
Environment.Default.Password=mypassword

# Current database connection configuration
#-------------------------------------------------------------------
------------------
```

```
Environment.Default.DBCon.WebsydianExpress.Driver=net.sourceforge.jtds.
jdbc.Driver
Environment.Default.DBCon.WebsydianExpress.DataSource=jdbc:jtds:sqlserv
er://localhost:1433/WebsydianExpress30
Environment.Default.DBCon.WebsydianExpress.User=myusername
Environment.Default.DBCon.WebsydianExpress.Password=mypassword
```

Configure express30 project.

a. Add your java project to express30 project.
Right-click on express30 → Select **Properties** → Select **Deployment Assembly** → Click **Add** → Select **Project** → Click **Next** > → Select your java project → Click Finish.

b. Configure web.xml file under express30/WebContent/WEB-INF. Open your web.xml with text editor and modify it so that it looks like the following.

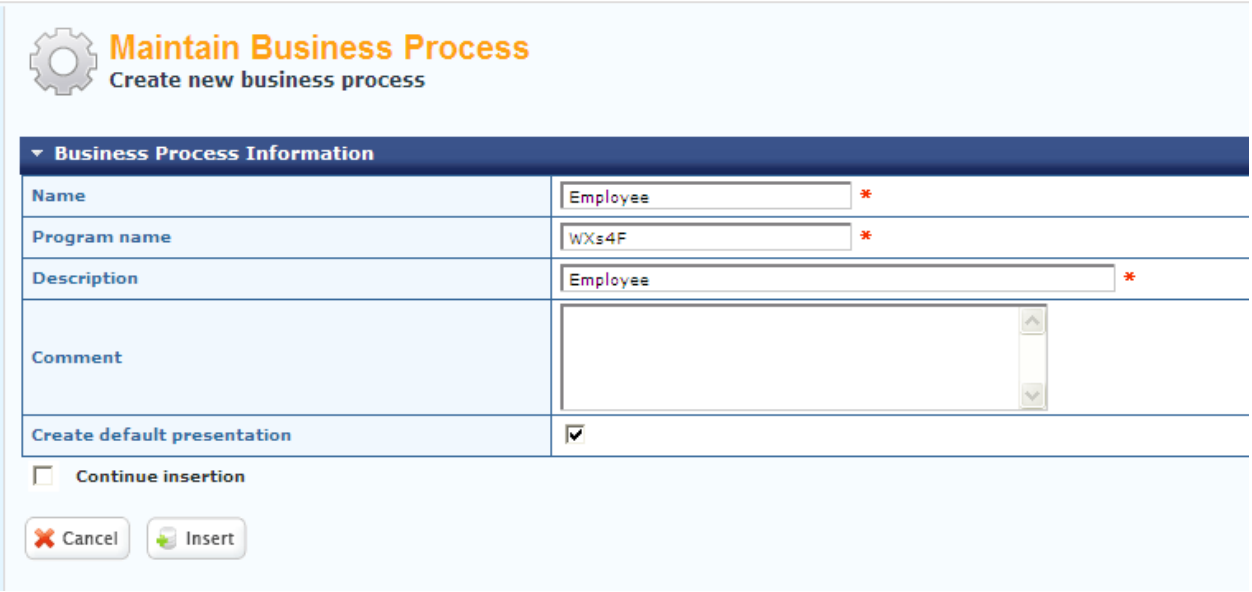
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp_ID">
<display-name>WebsydianExpress v3.0 for Java</display-name>
<description>WebsydianExpress v3.0 for Java</description>
<servlet id="Servlet_1">
     <servlet-name>WebsydianExpress30</servlet-name>
     <servlet-class>com.adcaustin.WebClient.servlet.WxpProxy</servlet-
class>
     <init-param>
           <param-name>servlet.propertyfile</param-name>
           <param-value></param-value>
     </init-param>
</servlet>
<servlet>
     <servlet-name>WebClientServlet</servlet-name>
     <display-name>WebClientServlet</display-name>
     <description></description>
     <servlet-class>
     com.adcaustin.WebClient.servletproxy.WebClientServlet</servlet-
class>
</servlet>
<servlet-mapping>
     <servlet-name>WebsydianExpress30</servlet-name>
     <url-pattern>/site/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
     <servlet-name>WebClientServlet</servlet-name>
     <url-pattern>/WebClientServlet</url-pattern>
</servlet-mapping>
</web-app>
```

c. Copy your images folder to express30/WebContent

Configure Websydian Express Admin site.

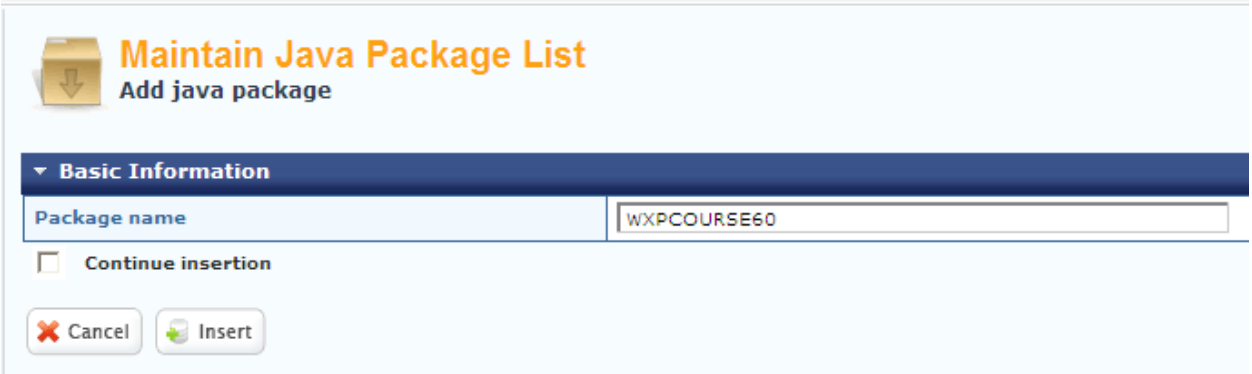
a. Go to http://localhost:8080/express30/site/basicsite/admin
Login as username: WSADMIN and password: ADMIN

b. Add business process
Expand **Content loaders** → Select **Business Processes** → Click **Insert** → Enter Name: myProgram, Program name:PlexImplementationName, and description – Create default presentation should be checked → Click **Insert**.

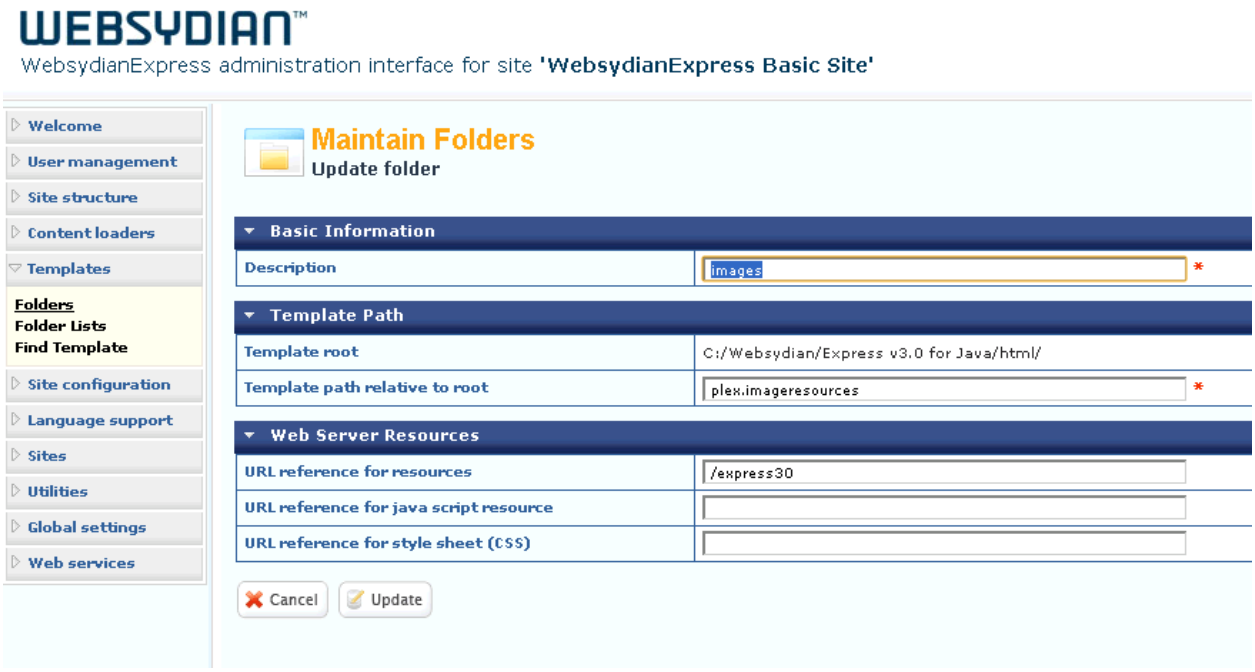


c. Add the business process to the site structure
Expand **Site structure** → Select **site structure** → Select **MENU** – Menu Loader → Click **Add** button → Select **Create a new site element** and select **business process** for Type of Site Element → Click **Next** → Select your business process → Enter Menu item text, alignment, sequence and display in frame (Please refer to Websydian Express help for details) → Select **Next** → Select the roles that can access the site element → Click **Finish**.

d. Add Java Packages.
Expand **Global settings** → Select **Java Packages** → Click **Insert** → Enter **package name** → Select **Insert**.



e. Set the path for images.
Expand **Template** → Select **Folders** → Click **Insert** → Enter the fields as follows.



Run CM WebClient application

Go to http://localhost:8080/express30/site/basicsite
Based on your application setting, you either need to login to run the application or run it anonymously.

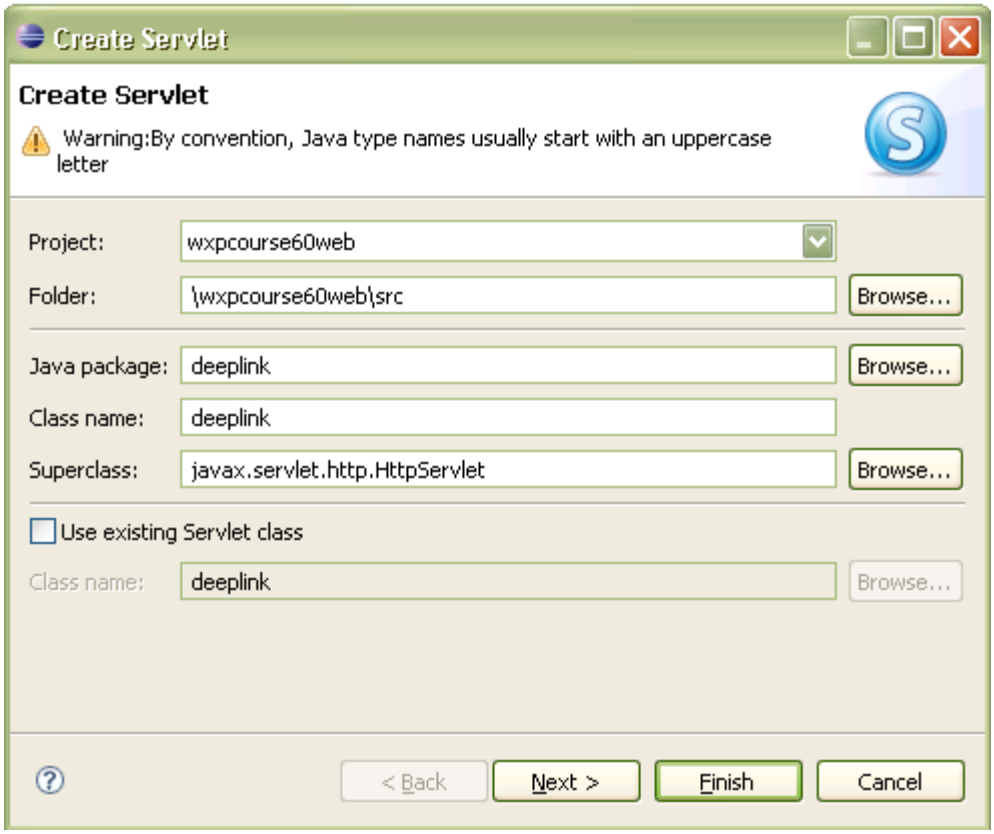
6. Calling WebClient from an External Application

This section demonstrates how to pass a parameter to WebClient function. The following example is based on wxpcourse60 model calling the function WXwF, which prompts the user to insert an item record.

The first step is to create a URL which calls the insert item function, without parameters. To accomplish this, open WebClient.properties file and add the following property.
`webclient.entry.url.InsertItem=WXPCOURSE60.WXwF`

Now, the servlet will respond to the URL `/WebClientServlet/InsertItem` by calling the function WXPCOURSE60.WXwF as the WebClient entry point. However, it does not yet allow parameters to be passed.

To pass parameters in, a custom servlet needs to be created. Open J2EE perspective. Right-click on web project → Select **New** → Select **Servlet**.
Enter *deeplink* as the Java package and Class name. For the servlet mapping, specify *InsertItem*. Click **Finish**.



In the WebClient documentation folder, there exists a sample custom servlet, called

deeplink.java, which will help you create a servlet that sends parameters to a Plex function. Copy and paste the code from deeplink.java to the newly created servlet. Change the package name to deeplink.

```
package deeplink;

import java.io.IOException;
```

Change the class name to deeplink.

```
public class deeplink extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet {

    private String encodeToXML(String str)
    {
        StringBuilder res = new StringBuilder();
        for (int i=0; i<str.length(); i++) {
            int c = str.charAt(i);
            if (c == '<')
                res.append("&lt;");
            else if (c == '>')
                res.append("&gt;");
            else if (c == '&')
                res.append("&amp;");
            else if (c < 32 || c >= 127)
                res.append("&#x" + Integer.toHexString(c) + ";");
            else
                res.append((char) c);
        }
        return res.toString();
    }
```

The parameters are passed to WebClient using an XML format. Locate the variable plexXML in the source code; this contains the XML data that is sent. Modify the input variable name and field name to match those expected by the WXwF function.

```
final String plexXML = "<PlexFunction
xmlns=\"http://adcaustin.com/ws-plex/1.0\">
<InputVariables Name=\"Input\"><FieldName=\"ItemID\">"+
encodeToXML(myVariable) +
"</Field></InputVariables></PlexFunction>";
```

Modify webClient URL from `final String webclientURL = "/webclient/test";` to `final String webclientURL = "/WebClientServlet/InsertItem";`

Now the InsertItem servlet will forward its requests to the WebClient servlet. Save changes.
The last change that needs to be made is the servlet mapping. It needs to have the characters /* at the end to match against any parameter that is passed in the URL. Open web.xml and change deeplink servlet mapping to

```
<url-pattern>/InsertItem/*</url-pattern>
```

And WebClientServletservlet mapping to

```
<url-pattern>/WebClientServlet/*<url-pattern>
```

The web.xml file should look like below.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">

  <display-name>belharraweb</display-name>

  <welcome-file-list>

    <welcome-file>index.html</welcome-file>

    <welcome-file>index.htm</welcome-file>

    <welcome-file>index.jsp</welcome-file>

    <welcome-file>default.html</welcome-file>

    <welcome-file>default.htm</welcome-file>

    <welcome-file>default.jsp</welcome-file>

  </welcome-file-list>

  <servlet>

    <description></description>

    <display-name>WebClientServlet</display-name>

    <servlet-name>WebClientServlet</servlet-name>

    <servlet-
class>com.adcaustin.webclient.servletproxy.WebClientServlet</servlet-
class>

  </servlet>

  <servlet-mapping>

    <servlet-name>WebClientServlet</servlet-name>

    <url-pattern>/WebClientServlet/*</url-pattern>

  </servlet-mapping>

  <servlet>

    <description></description>

    <display-name>deeplink</display-name>

    <servlet-name>deeplink</servlet-name>

    <servlet-class>deeplink.deeplink</servlet-class>

  </servlet>
```

```
    <servlet-mapping>
      <servlet-name>deeplink</servlet-name>
      <url-pattern>/InsertItem/*</url-pattern>
    </servlet-mapping>
  </web-app>
```

Save changes.
Build the web project and publish to the server.
To call the function, open a web browser and enter the following URL.
http://localhost:8080/wxpcourse60web/InsertItem/001
URL format:
http://<host>/<Context Root>/<Servlet Mapping>/<parameter>

7. Configuring Session Timeout in Apache Tomcat

The session timeout is the maximum length of time that Tomcat will retain the information pertaining to a user's session, in the absence of any user activity. This value is configurable per application as follows:

Open web.xml. Add the following XML element:

```
<listener>
  <listener-class>
    com.adcaustin.WebClient.servletproxy.WebSessionListener
  </listener-class>
</listener>
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

The code above will set the session to time out in 30 minutes. Save changes.

Open WebClient.properties and find webclient.entry.timeout.
Specify the name of the function that you want to display when the session is timed out.

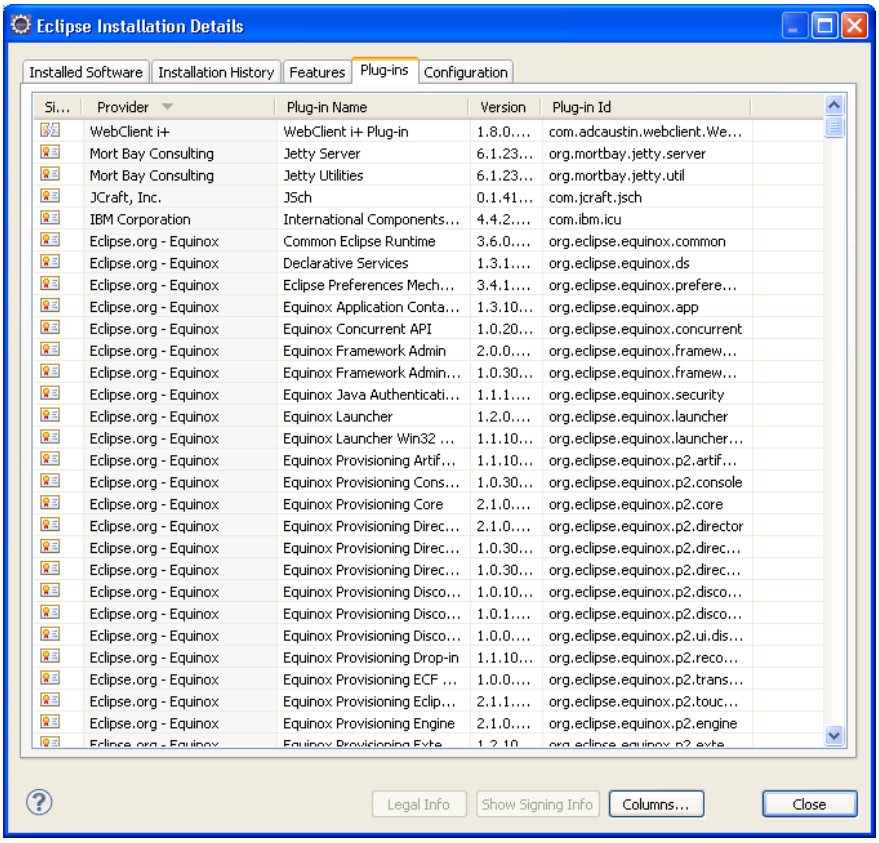
Save changes.

8. Upgrading CM WebClient v1.6.x to CM WebClient v1.8.x

a. Download com.adcaustin.webclient.WebClientPlugin-1.8.jar.

b. Close your eclipse application. Delete the old version of your WebClient Plugin Builder from eclipse/plugin folder and copy the new one that folder.

c. To check whether the Plugin Builder is installed properly.
Help → **About Eclipse** → Click on **Installation Details** → Sort *Plug-in Id* column in ascending order


Eclipse Installation Details
Installed Software
Installation History
Features
Plug-ins
Configuration
Si...
Provider
Plug-in Name
Version
Plug-in Id
WebClient i+ | WebClient i+ Plug-in | 1.8.0.... | com.adcaustin.webclient.We...
Mort Bay Consulting | Jetty Server | 6.1.23... | org.mortbay.jetty.server
Mort Bay Consulting | Jetty Utilities | 6.1.23... | org.mortbay.jetty.util
JCraft, Inc. | JSch | 0.1.41... | com.jcraft.jsch
IBM Corporation | International Components... | 4.4.2.... | com.ibm.icu
Eclipse.org - Equinox | Common Eclipse Runtime | 3.6.0.... | org.eclipse.equinox.common
Eclipse.org - Equinox | Declarative Services | 1.3.1.... | org.eclipse.equinox.ds
Eclipse.org - Equinox | Eclipse Preferences Mech... | 3.4.1.... | org.eclipse.equinox.prefere...
Eclipse.org - Equinox | Equinox Application Conta... | 1.3.10... | org.eclipse.equinox.app
Eclipse.org - Equinox | Equinox Concurrent API | 1.0.20... | org.eclipse.equinox.concurrent
Eclipse.org - Equinox | Equinox Framework Admin | 2.0.0.... | org.eclipse.equinox.framew...
Eclipse.org - Equinox | Equinox Framework Admin... | 1.0.30... | org.eclipse.equinox.framew...
Eclipse.org - Equinox | Equinox Java Authenticati... | 1.1.1.... | org.eclipse.equinox.security
Eclipse.org - Equinox | Equinox Launcher | 1.2.0.... | org.eclipse.equinox.launcher
Eclipse.org - Equinox | Equinox Launcher Win32 ... | 1.1.10... | org.eclipse.equinox.launcher...
Eclipse.org - Equinox | Equinox Provisioning Artif... | 1.1.10... | org.eclipse.equinox.p2.artif...
Eclipse.org - Equinox | Equinox Provisioning Cons... | 1.0.30... | org.eclipse.equinox.p2.console
Eclipse.org - Equinox | Equinox Provisioning Core | 2.1.0.... | org.eclipse.equinox.p2.core
Eclipse.org - Equinox | Equinox Provisioning Direc... | 2.1.0.... | org.eclipse.equinox.p2.director
Eclipse.org - Equinox | Equinox Provisioning Direc... | 1.0.30... | org.eclipse.equinox.p2.direc...
Eclipse.org - Equinox | Equinox Provisioning Direc... | 1.0.30... | org.eclipse.equinox.p2.direc...
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.10... | org.eclipse.equinox.p2.disco...
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.1.... | org.eclipse.equinox.p2.disco...
Eclipse.org - Equinox | Equinox Provisioning Disco... | 1.0.0.... | org.eclipse.equinox.p2.ui.dis...
Eclipse.org - Equinox | Equinox Provisioning Drop-in | 1.1.10... | org.eclipse.equinox.p2.reco...
Eclipse.org - Equinox | Equinox Provisioning ECF ... | 1.0.0.... | org.eclipse.equinox.p2.trans...
Eclipse.org - Equinox | Equinox Provisioning Eclip... | 2.1.1.... | org.eclipse.equinox.p2.touc...
Eclipse.org - Equinox | Equinox Provisioning Engine | 2.1.0.... | org.eclipse.equinox.p2.engine
Legal Info
Show Signing Info
Columns...
Close

d. Download WebClient-1.8.zip. Do not need to extract.

e. Open your workspace with Eclipse.

f. Close WebClient project in your workspace.
g. Delete WebClient project from your workspace and directory.

h. Import WebClient 1.8.0.
File → **Import** → select **Existing Projects into Workspace** → Select archive file and browse to the directory where you save your download.

i. Click **Finish**.

9. Customizing in CM WebClient

Attach point

A template for a given screen can be generated from more than one template by using attach points. The templates are arranged in order from root to leaf. An attach point is a location defined in a template that other templates further towards the leaf can populate with data. See Template generation.
Attach points can be defined in the root template or a page template using the `/(!AttachPoint)` command. When an attach point is defined, it is initally empty (it will not expand to anything), until it is populated by templates further down the inheritance path, or by controls via their control name property.
See Also `/(!AttachPoint)`

Bean controls

Some controls are implemented in Plex as a Java Bean control. Although Java Beans cannot be loaded in WebClient, their functionality can be emulated through the use of a custom template and customized source code objects in Plex. The name property of the bean control is the default name of template that is used (without the `.ctrl` extension.)
In the source code objects, variables in the control may be created using names which you define. The methods that do this are called setVar and getVar and they are located in the class WebBeanData. The source code objects may also call the method processAction to send a control action to the template.
In the bean control's template, the variables may be retrieved using the `/(!Var)` control command.
If the bean control sends a value to the server as a request parameter, the data in the parameter is taken to be a JSON object, and bean's variables are updated with the keys and values of that object.
Example: See `DojoTab.ctrl` in the distribution for an example of a tab control implemented in Javascript.

Control action

If the value of a control changes, the rendering on the web browser would also need to be updated. CM WebClient supports this two different ways. In the first method, page refresh, when the value of a control changes, the web browser is instructed to reload the page from the server. This will cause the template to be re-evaluated and the user will see the new data. This is the default method that is used to update the display.
In order to avoid page reloads, a second method is supported. In this method, called action updates, the page is not reloaded. Instead, the server sends back commands corresponding to only the changes that have occurred. These are called actions. An action has a name and an optional argument. Each action is expanded into a block of Javascript code that is then used to incrementally update the screen.
In order to use action updates with a control, the desired actions must be enabled for that control. This is accomplished by using the `/(!EnableAction)` template command within the control template for that control. The names of the actions that are going to be handled by the template are specified as parameters to that command. Then the `/(!Action)` command can be used to specifiy that commands that should be expanded when that action occurs. Ordinarily, the `/(!Action)` command is placed within an attach point that is within a Javascript output path.
The following actions are defined:

`SetValue`
: sent when the value of a field has been changed.

`SetState`
: sent when the state of the control has been changed.

`SetFocus`
: sent when the control is to be given focus.

`Refresh`
sent to a panel or site control when the panel needs to be refreshed. The argument is the panel control.

`Alert`
sent when a Dialog Message or Status Message has been sent to the panel.

`Enquiry`
sent when an Enquiry Message has been sent to the panel.

`Load`
sent to a site control when the child panel has been loaded. The argument is the panel control.

`Hide`
sent to a site control when the child panel has been hidden.

`Show`
sent to a site control when the child panel has been shown.

`Grid.Insert`
sent to a grid when a row has been inserted. The argument is the row.

`Grid.Clear`
sent to a grid when the grid has been cleared.

`Grid.Remove`
sent to a grid when a row has been removed. The argument is the row.

`Grid.Update`
sent to a grid when a row has been updated. The argument is the row.

Control context

When the template generator is in a control context, it is expanding a template for a specific control. This means it is possible to use control commands to access that control.
The template generator is automatically in a control context when it is expanding a control template. It is also in a control context when expanding the `/(FOCUS)` command.

Control name property

The control name property is a panel property that can be set in Plex for each control. Although it is not required to be set, it is strongly recommended that the control name be set for each control that is to be used under WebClient.
The control name property can take two syntaxes. The first syntax is simply a unique name given to the control. This name appears in the generated HTML and Javascript and aids in debugging. In the second syntax, the unique name is followed by a colon, then an attach point name, then optionally another colon

and template parameters. In this case, the control is given a name and also a location in the template where it is to appear. The control will be placed in the named attach point, which must be defined in the current panel's root template or page template. Any additional parameters following the attach point are passed to the control's template.
Example: If you have defined an attach point named `ButtonsArea`, and you want to place a button in it, you might assign the button the control name `ExitButton:ButtonsArea`.

Control reference

Certain template commands are not intended to expand to text, but rather are used to refer to a control to which another command is addressed. The next parameter of the command is taken to be the next command to be applied.
Any control with a control name can be used as a control reference, by giving the control name as a command. In addition, there are special template commands which are useful as control references.
Example: If you have a control named Grid1 and wish to access its size, you can may write `/(!Grid1:Size)`. Here, Grid1 is functioning as a control reference command. If you have a column heading control and need the NameID of the corresponding edit control, you can write `/(!Edit:!NameID)`. `/(!Edit)` is a control reference command.

Control template

A control template is used to produce HTML and Javascript for a control that is on a Plex panel. A control template is expanded and placed in the attach point that is associated with the control via the control name property.
The default control template that is used is in a file that is named by taking the type for of the control and prefixing it by Web. The extension that is used is `.ctrl`. For example, the default control template used for edit fields is `WebEdit.ctrl`. These control files are normally located in the SysTemplates folder.
To override and specify a specific control template that is to be used for a control, specify the `template=` parameter in the control name property for that control. The value of this parameter will be suffixed with the extension `.ctrl` and used instead of the default control template.
Example: If you have a custom template called `ToggleButton.ctrl` and wish to use this template for a check box control, you may set the control name parameter `template=ToggleButton`.

Default attach points for controls

If the control name property for a control does not specify an attach point for the control, a default attach point is selected based on the type of the control. The attach point is selected as follows:

Column heading
`ColumnsArea`
Grid edit control
`CellsArea`
Menu bar
`MenuBarArea`
Menu
`MenuArea`
Menu item or separator
`MenuItemArea`
Toolbar
`ToolbarArea`
Toolbar Button
`ToolArea`
All other controls
`MainArea`

In addition, the parameter `default` is passed to the template when the default attach point is selected.
Example: If an edit field control is given a control name property of `MyControl`, the default attach point of `MainArea` applies and the parameter `default` is also passed to the template. In other words, it is as if the edit field control had a control name property of `MyControl:MainArea:default`.

Name identifier

If a Plex panel contains child panels, the control name of a control, as given by `/(!Name)`, may not uniquely identify a control in the web browser. This is because even though the control is unique within a page, it still may appear in multiple child panels and create duplicates.
To resolve this issue, use the `/(!NameID)` command to generate a unique name identifier. This name will be prefixed by both the site's control name and the child panel's name. This name is suitable for use an identifier in the web browser's DOM.

Output path

CM WebClient can send a response back to the client of several different types. The output path is a parameter that determines what type of response is sent. In addition, some response formats can include hierarchical data. The output path can be used to index into that data.

The main output paths supported are:

`html`

the response mime-type is `text/html`. The contents are escaped according to HTML rules.

`json`

the response mime-type is `application/json`. The contents are encoded as a valid JSON expression.

For the json output type, a hierarchical naming scheme is possible. After the json component, which must be the top-level component in the path, a dot-separated path may be placed. The components of this path are created as JSON sub-objects in the JSON path.

Example: Suppose a response is created with the command `/(!OutTo:json.x.y.z)abc/(!OutTo)`. If this response is returned to a web browser and loaded as a JSON expression and assigned to the variable `t`, then `t.x.y.z == "abc"`.

Page template

A page template is an optional template that is expanded during the template generation process. A page template can place data into attach points that are defined by either the root template or another page template.
See Also: Template generation

Physical event

The Plex physical events which may be handled in WebClient are:

- Pressed
- Load Grid
- Double Click
- Select
- Mouse Down
- Mouse Move
- Mouse Up
- Gained Focus
- Lost Focus
- Changed
- Updated
- Select Menu
- Query Close
- Modified

See also: `/(!Event)`

Property modifier

A property modifier is an optional parameter which may be placed after the property name parameter in a `/(!Property)` command. The following property modifiers are available:

`x`

for a property value which is a coordinate in the format `x y`, expands to `x`.

`y`

for a property value which is a coordinate in the format `x y`, expands to `y`.

`hexcolor`

for a property value which is a color in the format `r g b`, expands to the color in HTML `#rrggbb` notation.

`htmlaccel`

for a property value which uses the & character to indicate that the next character is to be underlined, converts this notation to HTML using the `<u>` tag.

For example: control name is set to

`buttonA:MainArea:default:htmlaccel=&Apply`

Apply button can be executed by pressing CTRL+A

`htmlbr`

for a property value which uses the ^ character to indicate a line break, converts this notation to HTML using the `<br>` tag.

`nbspifempty`

if the property value is blank or consists only of spaces, expands to ` ` instead (as HTML source). Otherwise, leaves the value as is.

`html`

marks that the value is to be treated as HTML source.

`js`

marks that the value is to be treated as JavaScript source. This is the default option; if `html` is not specified, the property command will generate JavaScript source.

Root template

The root template is first template that is expanded during the template generation process. The default root template is called `WebShell-root.wcli`. It expands to the basic HTML and Javascript templates where all other template data is placed, via page templates and control templates.
See Also: Template generation

Source

Text that is expanded into the template can be of two types, source or data. By default, any text that is placed into a template at run-time is considered data. When text is considered data, it cannot contain markup commands such as HTML tags or Javascript expressions. If the data contains any special characters that would have meaning to the HTML or Javascript parser, such as angle brackets, ampersands, or quote markers, they will be automatically escaped appropriately for those parsers.
Specifically, if text is handled as data, and the output path is set to an HTML fragment, the characters less-than, greater-than, and ampersand are escaped with entity references. For example, `<foo>` is encoded as `<foo>`
If text is handled as data, and the output path is a Javascript fragment, the text is encoded as a valid Javascript string literal constant. This includes all quotation marks automatically. For example, `abc"def` is encoded as `"abc\"def"`.
If you have some text that is to be expanded at run-time and interpreted as an actual HTML tag or Javascript expression, with escaping, you must inform WebClient to handle it as source. This can be accomplished with a property modifier parameter on the property command.

Template generation

When generating a template for a panel, WebClient first attempts to locate a root template. The root template is a template available in the classpath such that the following two conditions are met:

- The name of the file consists of the implementation name of a function followed by `-root.wcli`.
- That function is in the inheritance path of the panel function being generated.

When there are two or more such functions in the inheritance path, the lowest (closest to the generated function) one takes precedence. If no root templates are available, template generation aborts with an error.

WebClient loads the root template, then it searches for page templates which may occur in the inheritance path between the generated function and the root template function. The following must be true for a valid page template:

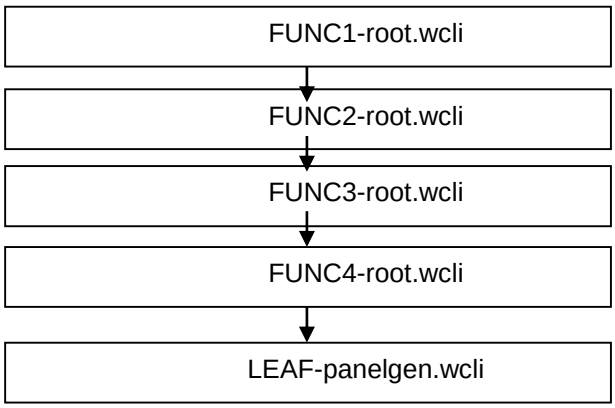
- The name of the template file consists of the implementation name of a function followed by `-page.wcli`.

- That function is in the inheritance path of the panel function being generated, and is lower than the root template that is in use.

The page templates are applied in order starting from the page template closest to the root template, and ending with the template closest to the panel function. Page templates are optional.

Finally, WebClient automatically generates a template for the panel function itself. This file does not need to be supplied by the user and is named by taking the implementation name of the panel function and appending `-panelgen.wcli`. This file is obtained by automatically expanding all of the control templates in the panel.

The following figure illustrates the process:



10.Servlet 3.0 Template Resource Deployment

Introduction

CM WebClient now makes it easier to deploy your static HTML, CSS, and JavaScript files along with your templates. This is done by following the Java EE Servlet 3.0 specification. This method of deployment is supported even if your web container does not support Servlet 3.0.

Web Library ID

CM WebClient defines an identifier known as a **web library ID**. This is a string value that differentiates the static resources belonging to different modules within the workspace.

In this configuration, page templates and control templates should be located in the src folder of a Java project, and static resources should be located in the `META-INF/resources` folder of the project.

Additionally, the library ID should be set in the file `META-INF/MANIFEST.MF`:

```
Web-Library-Id: <package name>
```

The package name can be any valid, unique name.

Now, the /(!WsRes) template command in the template files will refer to a directory that contains the contents of the `META-INF/resources` folder at run-time.

11. How to Customize CM WebClient StyleSheet

1. Create a new folder named *css* in your web project under **WebContent**.

2. Create a new css file named *mystyle.css* inside the newly created folder.

   ```
   body{  background-color: black;}
   ```

3. Create a new java project named *customTemplate* in your workspace.

4. Add *customTemplate* project to your java project build path.
   Right-click on your java project → Select **Properties** → Select **Java Build Path** → Open **Projects** tab → Click **Add…** button → Check *customTemplate* project → Click **OK**.

5. Create a new folder named *META-INF* under customTemplate/src folder

6. Create a new file in META-INF folder named *MANIFEST.MF*.

MANIFEST.MF should contain the following.
Web-Library-Id: com.mycompanyname.myprojectname

Note: Make sure there is a new line after Web-Library-Id:
com.mycompanyname.myprojectname

7. Create a new file named *customCSS-page.wcli* under customTemplate/src folder. customCSS-page.wcli should contain the following.
```
/(!CSSInit)
@import "/(!WsRes)/css/mystyle.css";
/(!CSSInit)
```

8. Open *WebClient.properties* and add the following property.
resurl.com.mycompanyname.myprojectname = /<contextRoot>
for example: `resurl.com.mycompanyname.myprojectname = /wxpcourse60web`

9. Create a function in your Plex model.
```
customCSS is a FNC WEBCLIENT/~WebUI
customCSS impl name customCSS
```
Your function that needs to have custom style sheet should inherit from customCSS triple in the order like below.
```
MyFunction is a FNC WEBCLIENT/~WebShell
MyFunction is a FNC customCSS
```
10. Generate and build your function and you should see that the screen background color is in black

12. Troubleshooting

Q: I am not able to connect to AS/400 in CM WebClient or I am getting java.io.UnsupportedEncodingException: Cp037. What should I do?

A: Make sure that your function works in Java Swing. If it is not working in Java Swing, check if your eclipse is using the correct JDK version. You should be using JDK 1.5 or higher.

Q: What are the steps to run Java Swing in Eclipse?

A: Go to **Run** → **Run Configurations**… → Select **Java Application** → Click **New launch configuration**.

In the **Main** tab window, browse and select your java project (the project that has your java source files). For the main class, search for *ObRun.ObPanel.ObLaunch* and click **Apply**.

Open **Arguments** tab window → In the Program arguments field, enter your function entry point in the format of <package name>.<function implementation name> "Path= " → Click Apply → Click Run.

For example: WXPCOURSE60.WXuF "Path= "

Q: What should I do if I am getting a java.lang.NullPointerException when generating the web template?

A: If you are using PlexResource folder to store your Plex .panelresource or .resource files, the PlexResource folder should set as a source folder. To set your PlexResource folder as a source folder:

1. Right-click on your java project and select **Properties**.
2. In the **Source** tab, click add folder and add the PlexResource folder to the source.

Q: In CM WebClient v1.4 we have to create a js folder for javascript files. Why there is no js folder in CM WebClient v1.6?

A: In order to ease our customer with the upgrade process, we have included the javascript files in WebClient project.

Q: How do I check if my projects are published correctly in Tomcat server?

A: Go to <workspace directory>/ .metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\<contextroot> \WEB-INF\lib

Q: Some browsers like Firefox and Safari do not set locale correctly. What can I do?

A: Create a page template, for example: locale-page.wcli on your custom template project. In your locale-page.wcli file, it should have something like the following.

```
/(!djConfig)
 locale:'fi-fi',
/(!djConfig)
```

Above is the example for setting locale in Finnish. Remember to add a new triple in your function to apply the new page template.
myFunction is a FNC ~WebShell customPage is a FNC ~WebUI
myFunction is a FNC customPage customPage impl name locale

Q: Subclipse does not work on WebClientLiveInstall. What should I do?

A: WebClientLiveInstall is not designed for CM WebClient development purpose. If you are setting up CM WebClient development environment, you should build your environment from scratch. Subclipse will work with official eclipse installation.

Q: What do I do if Eclipse console shows web template generator error like below

```
"WebClientTestModel.toolbar_ObFnc$toolbar_ObPnl:  [Show
traceback]WebClientTestModel.toolbar_ObFnc$toolbar_ObPnl.generateTempl
ate([B, com.adcaustin.util.AFile)" ?
```

A: You need to install CM WebClient 1.8 project and make sure that you have the same WebClientBuilderPlugin installed in your Eclipse plugin.

Appendix A: Template Commands

--Comment

Syntax: `/(!-- comment)`

Description: This command is a server-side comment. It has no effect.

ALERTS

Syntax: `/(ALERTS)`

```
...
/(ALERTS)
```

Description: The contents of the data inside the `/(ALERTS)` block is expanded once for every outstanding alert message associated with the panel. The details of each

message can be retrieved within the block by using the `/(AlertMsg)`, `/(AlertType)`, and /(AlertFdbk) commands.

Action

Syntax: `/(!Action:[[control action]])`

```
...
/(!Action)
```

Description: Allows a template to handle control actions. The action must have been previously enabled with the `/(!EnableAction)` command. The data inside of the `/(!Action)` command is expanded for every action of that type which has been sent to the command.

For an example, see the entry for `/(!EnableAction)`.

ActionArg

Syntax: `/(!ActionArg)`

Description: The command is valid within the body of an `/(!Action)` command. When specified, it expands to the argument of the control action being processed, if any. The value of the argument depends on the type of action. For more information, see the documentation specific for the control action.

AttachPoint

Syntax:

```
/(!AttachPoint:name=[[attach point]]
    [ :inlinetemplate ]
    [ :sep=separator]
        [ :args...])
[ ... inline template data ...
  /(!AttachPoint) ]
```

Description: Defines an attach point in the current template where future templates may populate. The attach point is identified by the given name. The `/(!AttachPoint)` command expands to an empty string; however, the position is marked and when future templates invoke the attach point, their output will be placed at that position. The attach point is invoked by using a command with the same name as the attach point.

If the `inlinetemplate` flag is set, the AttachPoint command accepts a block, and this block is expanded as a template whenever the attach point is used. If the inlinetemplate flag is not present, the AttachPoint command must not have a block.

If the `sep` parameter is specified, it specifies text that is used as a separator between invocations of the attach point. The separator text is placed in between every invocation.

Any other parameters that are specified become default arguments for the templates attached to the attach point.

ButtonGroup

Syntax: `/(!ButtonGroup)`

Description: Expands to the control name of the current button group. The button group is the control name that is associated with the current group of radio buttons.

ButtonValue (Radio Button)

Syntax: `/(!ButtonValue)`

Applies to: Radio button

Description: Expands to the button value of radio buttons.

CheckedValue (Checkbox)

Syntax: `/(!CheckedValue)`

Applies to: Checkbox

Description: Expands checked value of the check box field.

DataType

Syntax: `/(!DataType)`

Description: Expands to the data type of the field which is bound to the control. The value will be one of the following:

- `Char`
- `FixedDec`
- `Char`
- `Long`
- `Integer`
- `Double`

- Date
- Time
- Object

DecLength

Syntax: /(!DecLength)

Description: Expands to the design decimal places of the field which is bound to the control.

Edit

Syntax: /(!Edit)

Description: Restrict: WebColumnHeadingData .

References the column's associated edit control. This is a control reference command.

EditNameID

Syntax: /(!EditNameID)

Description: Expands to the fully qualified name identifier of the edit control that is associated with the current control, which must be a Column Heading control.

EditProperty

Syntax: /(!EditProperty:[[property name]] [:[[property modifier]]])

Description: Works similarly to /(!Property), except the properties are retrieved from the edit control that is associated with the current control, which must be a Column Heading control.

Embed

Syntax: /(Embed:output type [:cdata])

...

/(Embed)

Description: The contents of the Embed block are expanded and encoded with the specified output type. This is used when one output type is embedded in another; for example, JavaScript within HTML, or HTML within JavaScript.

The output type can be:

`html`

Encodes the output as HTML.

`xml`

Encodes the output as XML.

`js`

Encodes the output as a Javascript expression.

`plain`

The output is not encoded. Use with `cdata` (see below) to get out of HTML escaping.

If the flag `cdata` is specified, and the current output type is HTML, then the current HTML element is assumed to be of type CDATA, and entity encoding is bypassed as is necessary for CDATA elements.

Example: The following illustrates retrieving the value of MyField into a Javascript variable, using a script embedded within an HTML page:

```
<script type="text/javascript">
      /(Embed:js:cdata)
            var fld_value = /(MyField);
            ...
/(Embed)
</script>
```

The following illustrates generating a snippet of HTML from within a Javascript function:

```
  function my_function()  {
var html = "<span class='myclass'>"   +
/(Embed:html)/(MyField)/(Embed)  + "</span>";
...
}
```

EnableAction

Syntax: `/(!EnableAction:[[control action]] [ ,[[control action]]... ])`

Description: When specified for a control, indicates that the template will handle the given control actions. Any number of actions may be specified, separated by commas. Once an action is enabled, the runtime will not automatically refresh the page when an action of that type is processed. Instead, it will enqueue the action to a list associated with the control. The template must then expand the action list by using the `/(!Action)` command.

Example: The following example shows how a read-only field can change its value without refreshing the screen.

```
/(!-- The field is mapped as an HTML SPAN element.)
<span id="/(!NameID)">
     /(!This)
</span>

/(!-- Indicate that when the SetValue control  action is processed, we
will handle it here.)
/(!EnableAction:SetValue)

/(!-- Place Javascript code inside of the JS attach point to perform
the update.)
/(!JS)

      /(!-- The code in the following block will be run  as Javascript
when the value of the    control  is changed.)
      /(!Action:SetValue)
           document.getElementById("/(!NameID)").innerText = /(!This);
      /(!Action)
/(!JS)
```

Event

Syntax: `/(!Event:phys=[[physical event]])`

Description: If the given physical event is mapped on this control, expands to its mapped event number. Otherwise, expands to the empty string.

Example: The following template will expand differently depending on whether or not the Pressed event is mapped to the current control.

```
      /(!If:/(!Event:phys=Pressed))
           The pressed event is mapped
      /(!Else)
      The pressed event is not mapped
   /(!If)
```

FOCUS

Syntax: `/(FOCUS)`
`...`
`/(FOCUS)`

Description: The contents of the data inside the `/(FOCUS)` block is expanded once for each control on the panel that has received initial focus. The block is expanded with a valid control context for each focused control.

FindControl

Syntax: `/(!FindControl: [[control id]])`

Description: This control reference command yields the control of the panel that has the specified Plex control ID. The control ID is a numeric value that is assigned by the Plex panel generator. It is unstable (changes with every generation) and is included in certain control properties, which is why you may need this command.

GridEnd (Grid)

Syntax: `/(!GridEnd)`

Applies to: Grid

Description: Expands to Y if the Plex program has executed Set Grid End On for this grid. Otherwise, expands to nothing.

If

Syntax:
```
/(!If:value)
...
[ /(!Else) ]
...
/(!If)
/(!If:value1=value2)
...
[ /(!Else) ]
...
/(!If)
```

Description: This command performs a comparison. It has two forms. In the first form, the contents of the `/(!If)` block are expanded if `value` expands to a non-empty string. In the second form, the contents are expanded if `value1` is equal to `value2`. In both forms, an optional `/(!Else)` clause may be specified. The `/(!Else)` block is terminated by the `/(!If)` command terminator; it does not have its own terminator.

IgnoreActions

Syntax: `/(!IgnoreActions)`

Description: When specified for a control, causes the control to ignore any control actions that are sent to it. This command always expands to the empty string.

This is useful for a template which does not respond to actions, to prevent the screen from refreshing.

ImplName

Syntax: `/(!ImplName)`

Description: Expands to the implementation name of the field which is bound to the control.

Length

Syntax: `/(!Length)`

Description: Expands to the design length of the field which is bound to the control.

NameID

Syntax: `/(!NameID)`

Description: Expands to the fully qualified name identifier of the current control. The NameID is unique even when child panels are used. (The control name itself in such a situation need not be unique.)

OptSelected (Combo Box)

Syntax: `/(!OptSelected:[[option]])`

Applies to: Combo Box

Description: Expands to Y if the option selected is the one indicated.

OptSelected (List)

Syntax: `/(!OptSelected:[[option]])`

Applies to: List

Description: Expands to Y if the option selected is the one indicated.

OptText (Combo Box)

Syntax: `/(!OptText)`

Applies to: Combo Box

Description: Returns the text of the current option of a combobox. This command is valid within the `/(!This)` command block of a combobox control.

OptText (List)

Syntax: `/(!OptText)`

Applies to: List

Description: Returns the text of the current option of a listbox. This command is valid within the `/(!This)` command block of a listbox control.

OptValue (Combo Box)

Syntax: `/(!OptValue)`

Applies to: Combo Box

Description: Returns the value of the current option of a combobox. This command is valid within the `/(!This)` command block of a combobox control.

OutTo

Syntax: `/(OutTo:[[output path]])`

```
....
      /(OutTo)
```

Description: The contents of the OutTo block are placed into the response stream at the specified output path. The OutTo command itself expands to nothing; all data is placed in the path.

Owner

Syntax: `/(!Owner)`

Description: Returns the owning panel.

Owner

Syntax: `/(!Owner)`

Description: References the panel that owns the current control. This is a control reference command.

Panel

Syntax: `/(!Panel)`

Description: This control reference command refers to the panel control. This is the root element of the control hierarchy.

Param

Syntax: `/(!Param:[[parameter name]])`

Description: Expands to the value of the given parameter in the current template.

If the current template is a control template, the parameter may be a control name parameter. If the current template is an inline template, the parameter may be a parameter specified in the inline template invocation.

Parent

Syntax: `/(!Parent)`

Description: References the parent of the control in the control hierarchy. This is a control reference command.

Property

Syntax: `/(!Property:[[property name]] [ :[[property modifier]] ] )`

Description: Expands to the value of the specified property of the current control. If the property is not defined, expands to the empty string.

An optional property modifier may be specified.

Render

Syntax: `/(!Render:[[output path]] [ :args... ] )`

Description: Performs rendering of the current panel, and sends the result of rendering the panel to the specified output path. If any optional arguments are specified, they are carried over as request parameters and may be retrieved with the `/(!Req)` command.

Site (Site control)

Syntax: `/(!Site)`

Applies to: Site control

Description: Expands to the panel object that is loaded in the current site control.

State

Syntax: `/(!State)`

Description: Expands to the current control's state value, as an integer.

This

Syntax: `/(!This)`

Description: Expands to the value of the current control. This command also works as a control reference, referring to the current control.

The behavior of this command depends on which type of control is being used. See the control-specific documentation for details.

This (Combo Box)

Syntax: `/(!This) ... /(!This)`

Applies to: Combo Box

Description: Expands to the contents of the block, repeated once for every option in the combo box control.

This (Grid)

Syntax: `/(!This) ... /(!This)`

Applies to: Grid

Description: Expands the contents of the command block once for every grid row in the grid.

This (List)

Syntax: `/(!This) ... /(!This)`

Applies to: List

Description: Expands to the contents of the block, repeated once for every option in the list box control.

Title

Syntax: `/(!Panel:!Title)`

Description: Expands to the title of the current panel.

Type

Syntax: `/(!Type)`

Description: Returns the type of the current control. This is the value from the Type property in the panelresource file.

UncheckedValue (Checkbox)

Syntax: `/(!UncheckedValue)`

Applies to: Checkbox

Description: Expands unchecked value of the check box field.

Visible

Syntax: `/(!Visible)`

Description: Expands to Y if the current panel is visible. Otherwise, expands to nothing.

WsRes

Syntax: `/(!WsRes)`

Description: At runtime, expands to the web resources path that associated with the web library id of the current template. The web library id is defined by the project containing the template; that id is then mapped to a URL resource path by a runtime-specific configuration mechanism. In WebClientServlet, the URL will map to the META-INF/resources folder of the project.

Example: The following HTML image element refers to an image which is located in the images directory under the current project's web resources path:

```
<img src="/(!WsRes)/images/myimage.png"></img>
```

WsResLib

Syntax: `/(!WsResLib:web library id)`

Description: At runtime, expands to the web resources path that associated with the given web library id. The web library id is then mapped to a URL resource path by a runtime-specific configuration mechanism. In WebClientServlet, the URL will map to the META-INF/resources folder of the project.

Example: The following HTML image element refers to an image which is located in the images directory under the web resources of the project with web library ID com.acme.lnf:

```
<img src="/(!WsResLib:com.acme.lnf)/images/myimage.png"></img>
```

getVar(String var)

Syntax: `getVar(String var)`

Applies to: Bean controls

Description: Retrieves the value of the specified bean variable in the current control.

jsonColVis (Grid)

Syntax: `/(!jsonColVis)`

Applies to: Grid

Description: Returns a list of the visibilities of each column on the grid, as a Javascript array. Each element is either true if the column is visible, or false if it is not.

jsonCols (Grid)

Syntax: `/(!jsonCols)`

Applies to: Grid

Description: Returns a list of the names of the columns of the grid, as a Javascript array.

jsonImplNames (Grid)

Syntax: `/(!jsonImplNames)`

Applies to: Grid

Description: Returns a list of the implementation names of the column fields of the grid, as a Javascript Array.

jsonModes (Grid)

Syntax: `/(!jsonModes)`

Applies to: Grid

Description: Returns a list of the read-only or edit modes of each column on the grid, as a Javascript array. Each element is either "Read Only" or "Edit".

jsonOptions (Combo Box)

Syntax: `/(!jsonOptions)`

Applies to: Combo Box

Description: Returns a list of the names of the options in the combo box, as a Javascript list.

jsonOptions (List)

Syntax: `/(!jsonOptions)`

Applies to: List

Description: Returns a list of the names of the options in the list box, as a Javascript list.

jsonRows (Grid)

Syntax: `/(!jsonRows)`

Applies to: Grid

Description: Returns the contents of the grid as a two dimensional array. The outer dimension is rows, the inner dimensions is columns.

jsonSelected (Grid)

Syntax: `/(!jsonSelected)`

Applies to: Grid

Description: Returns a list of row numbers in the grid that are selected. The rows are numbered started from zero.

jsonValues (Combo Box)

Syntax: `/(!jsonValues)`

Applies to: Combo Box

Description: Returns a list of the values of the options in the combo box control, as a Javascript list.

jsonValues (List)

Syntax: `/(!jsonValues)`

Applies to: List

Description: Returns a list of the values of the options in the list box control, as a Javascript list.

processAction(String actionName [, Object arg])

Syntax: `processAction(String actionName [, Object arg] )`

Description: Sends a control action with the specified name to the control. The action name can be matched the with `/(!Action)` control command. If the action argument `arg` is specified, it may be retrieved with the `/(!ActionArg)` control command.

setVar(String var, Object val)

Syntax: `setVar(String var, Object val)`

Applies to: Bean controls

Description: Sets the value of the specified bean variable in the current control to `val`.

value (Combo Box)

Syntax: `/(!value)`

Applies to: Combo Box

Description: Expands to the text value of the option that is currently selected.

value (List)

Syntax: `/(!value)`

Applies to: List

Description: Expands to the text value of the option that is currently selected.

jsonSitePanels

Syntax: `/(!jsonSitePanels)`

Applies to: Site

Description: Expands to an array of JSON objects, each of which represents a child panel which has been loaded into the site control. The structure of each JSON object is:

{ name: the name of the panel title,
pnlid: the id of the panel which can be used to activate or deactivate it,
active: a boolean flag which is true if the panel is the currently active one }

Appendix B: JDBC Driver Configuration

Configuring and Using SQL Server 2005

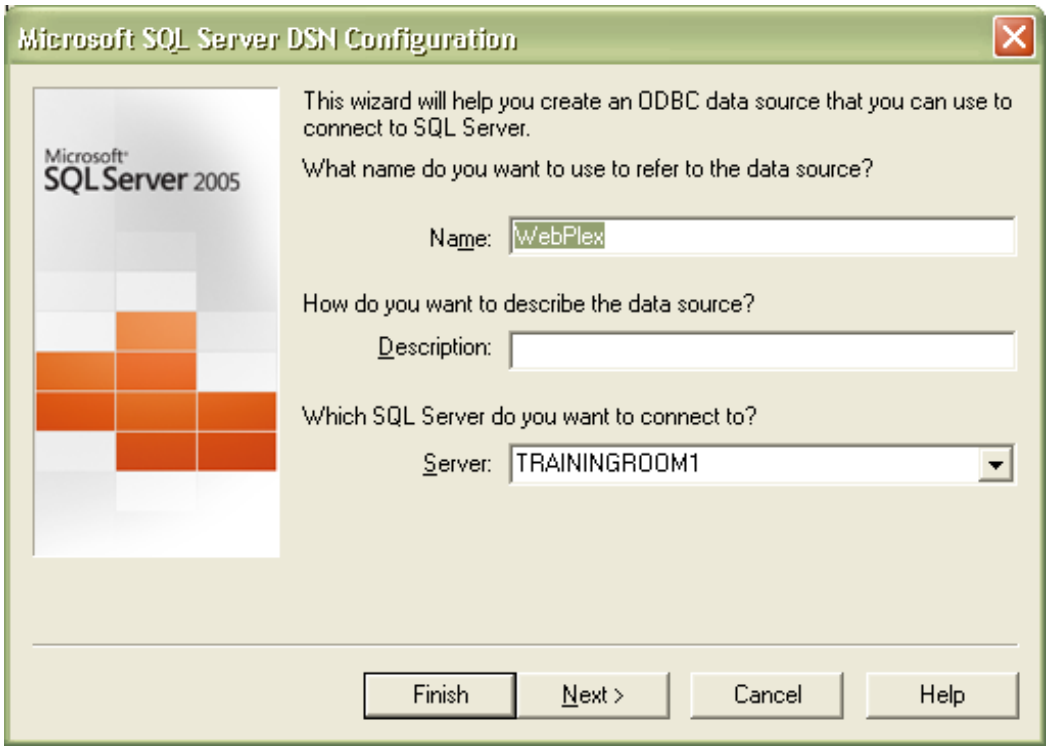
Download SQL Server 2005 JDBC driver at
http://www.microsoft.com/downloads/en/details.aspx?familyid=C47053EB-3B64-4794-950D-81E1EC91C1BA&displaylang=en

Creating an ODBC Connection for SQL Server 2005

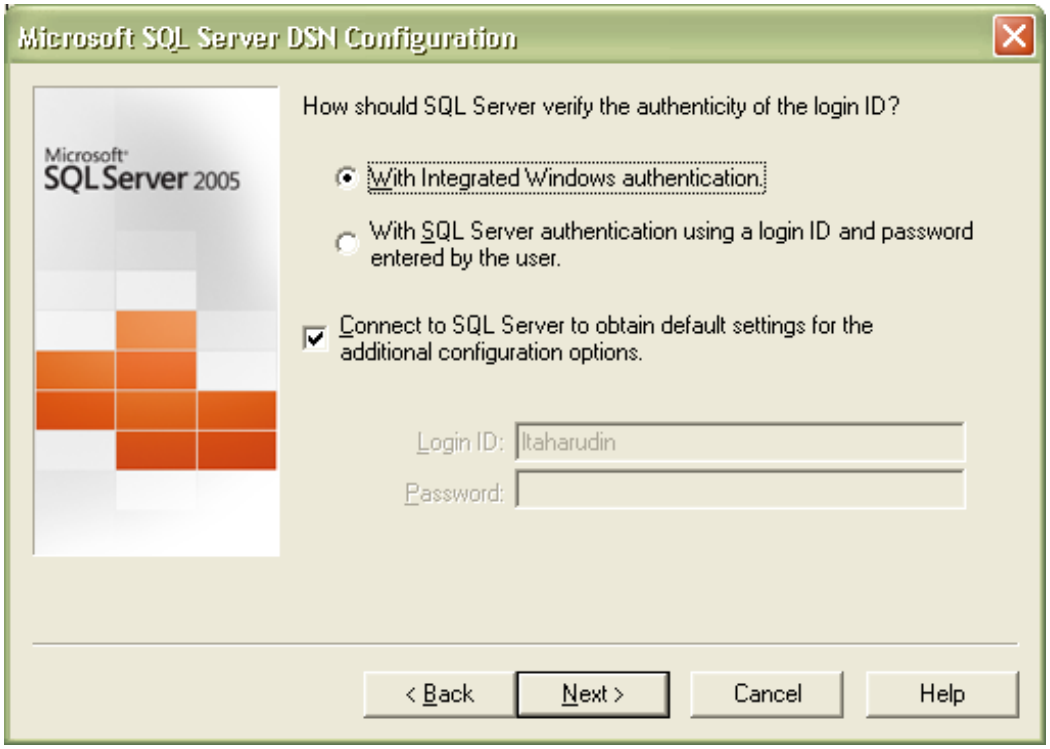
a. Go to **Control Panel → Administrative Tools → Data Sources (ODBC)**

b. Open **User DSN** tab → Click **Add...** → Select **SQL Native Client driver** → Click **Finish**.

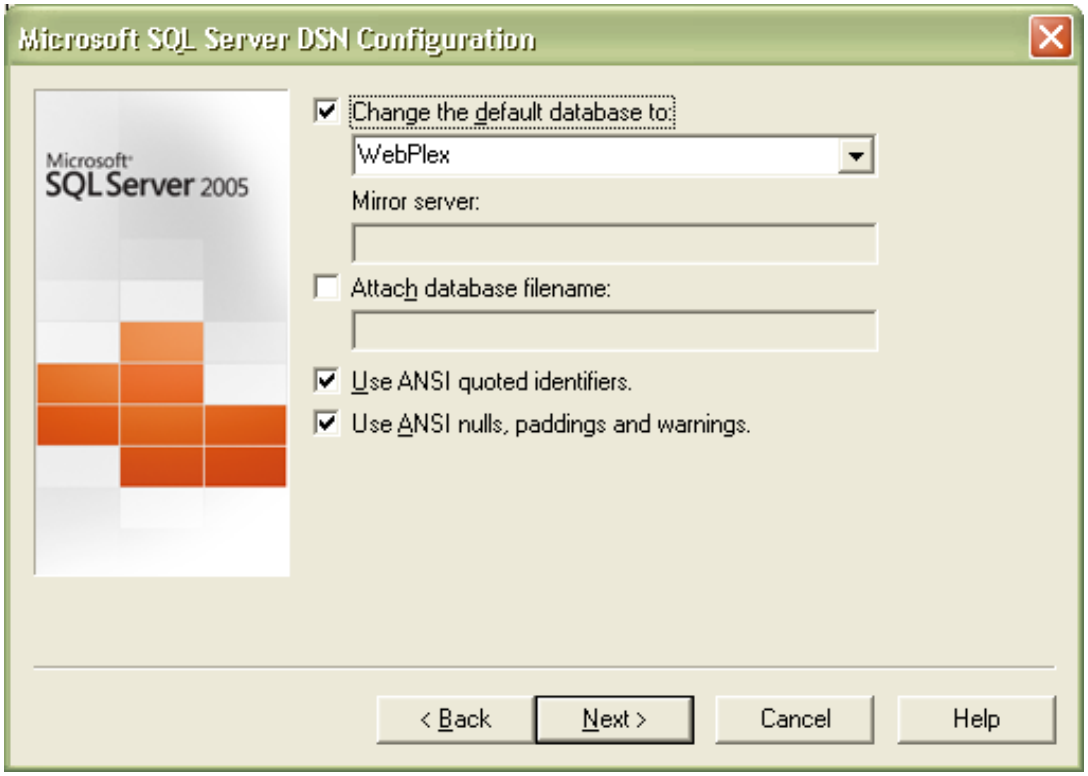
c. Enter data source information → Click **Next** >


Microsoft SQL Server DSN Configuration
Microsoft SQL Server 2005
This wizard will help you create an ODBC data source that you can use to connect to SQL Server.
What name do you want to use to refer to the data source?
Name: WebPlex
How do you want to describe the data source?
Description:
Which SQL Server do you want to connect to?
Server: TRAININGROOM1
Finish
Next >
Cancel
Help

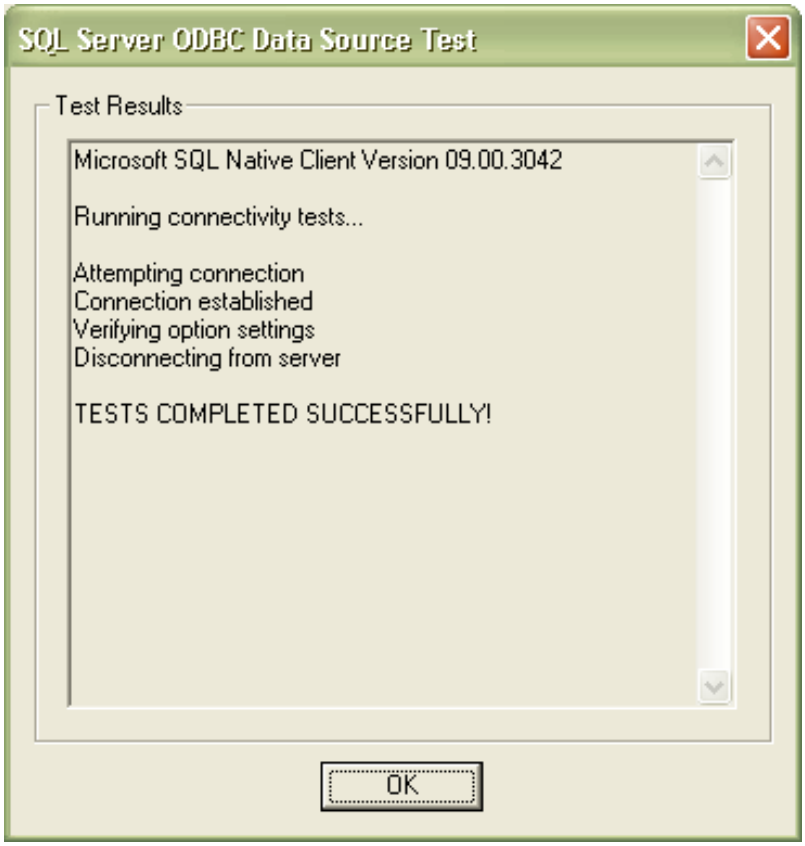
d. Select **"With Integrated Windows authentication."**


Microsoft SQL Server DSN Configuration
Microsoft SQL Server 2005
How should SQL Server verify the authenticity of the login ID?
With Integrated Windows authentication.
With SQL Server authentication using a login ID and password entered by the user.
Connect to SQL Server to obtain default settings for the additional configuration options.
Login ID: ltaharudin
Password:
< Back
Next >
Cancel
Help

e. Change the default database to your database name → Click **Next** >



f. Click **Finish** → Click **Test Data Source...**
It should display TESTS COMPLETED SUCCESSFULLY



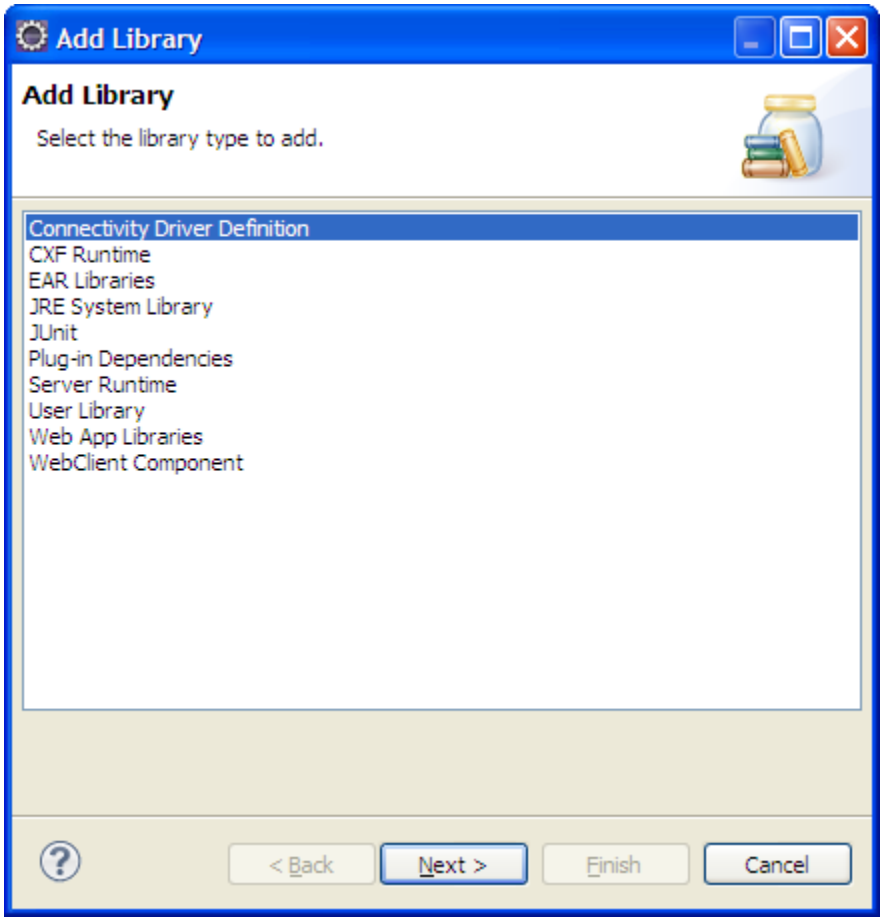
g. Click **OK** to complete the setup

Configuring the SQL Server 2005 JDBC Driver in Eclipse

a. Open obclient.properties and edit the following parameters. Enter the username and password to connect to the database.

```
Environment.Default.Driver=com.microsoft.sqlserver.jdbc.SQLSer
verDriver
Environment.Default.DataSource=jdbc:sqlserver://localhost:port
number;databasename=yourdbname
Environment.Default.User=username
Environment.Default.Password=password
```

b. Add the JDBC driver on the Java Project. Right-click on the java project → Select **Properties** → Open **Java Build Path** → Open **Libraries** tab → Click **Add Library...** → Select **Connectivity Driver Definition** → Click **Next** >



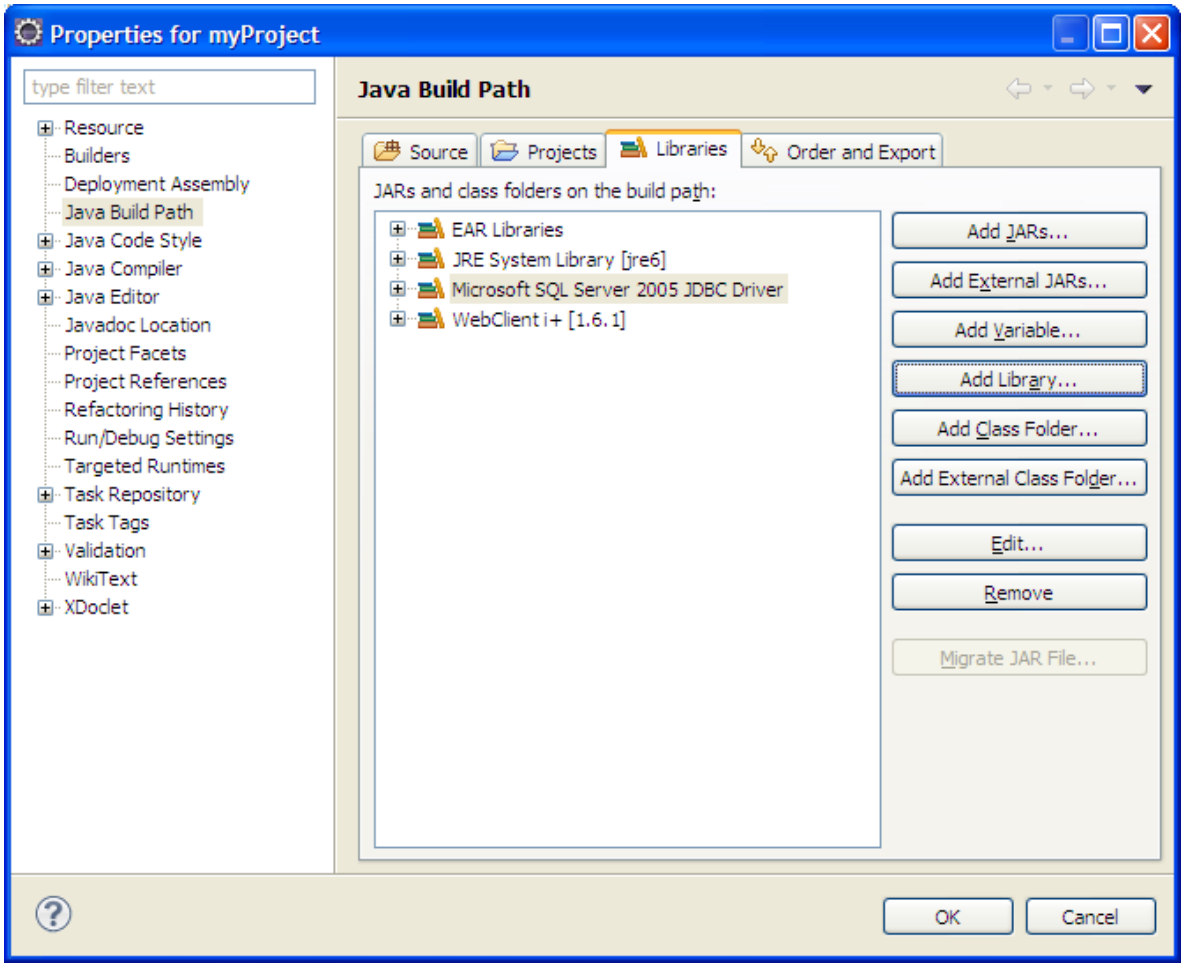
c. Click on **Add New Driver Definition** (an icon beside Available Driver Definitions combo box).

d. On the Name/Type tab select **Microsoft SQL Server 2005 JDBC Driver**.

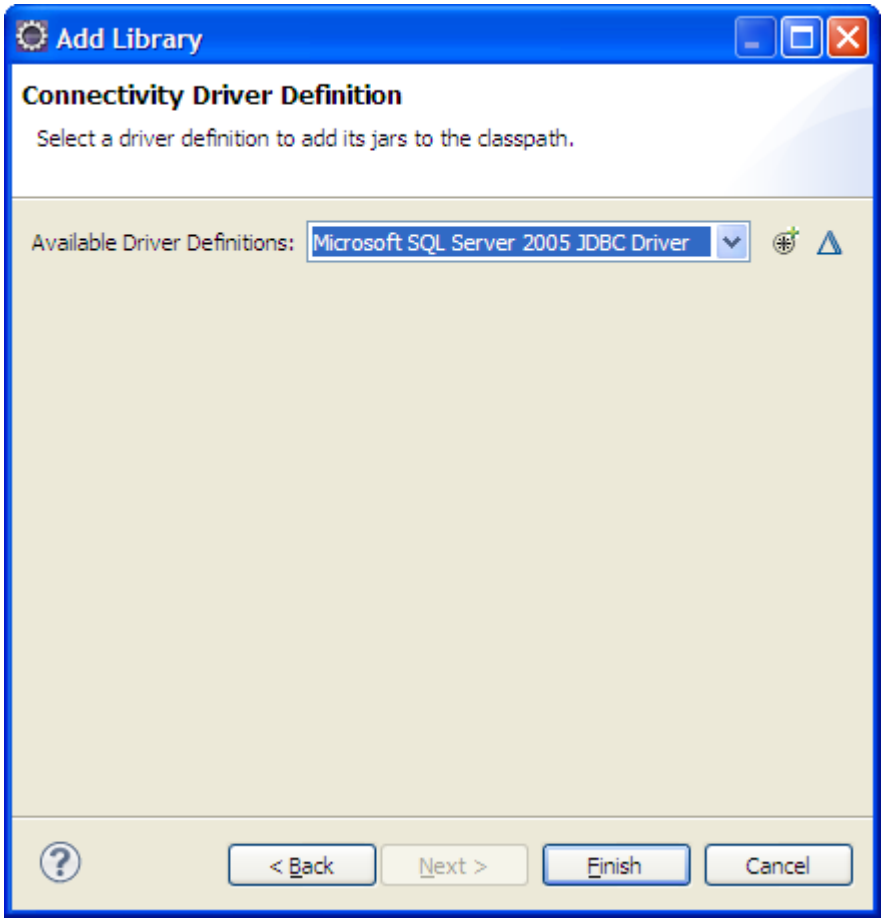
e. Open **JAR List** tab, click **Clear All** then click **Add JAR/Zip…** to select sqljdbc.jar from your sqljdbc.jar installed directory.

f. Open **Properties** tab, enter your connection URL, database name and password according to obclient.properties

g. Click **Finish**.



h. Add the JDBC drive to your web project.
Right-click on your web project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add Library…** → Select **Connectivity Driver Definition** → Click **Next** > Select **Microsoft SQL Server 2005 JDBC Driver** from Available Driver Definitions → Click **Finish**.



Configuring and Using MySQL Server

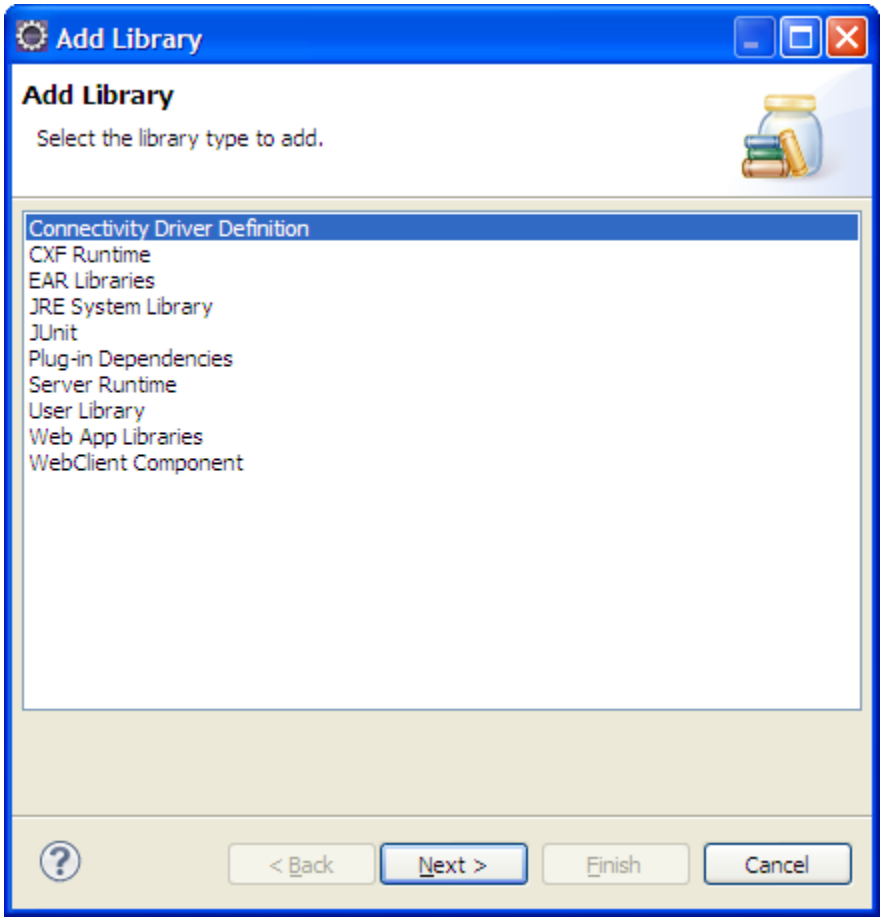
Download MySQL Connector/J
http://www.mysql.com/downloads/connector/j/
Refer to mySQL website for installation instruction

Configuring MySQL JDBC Driver in Eclipse

a. Open obclient.properties and edit the following parameters. Enter the username and password to connect to the database.

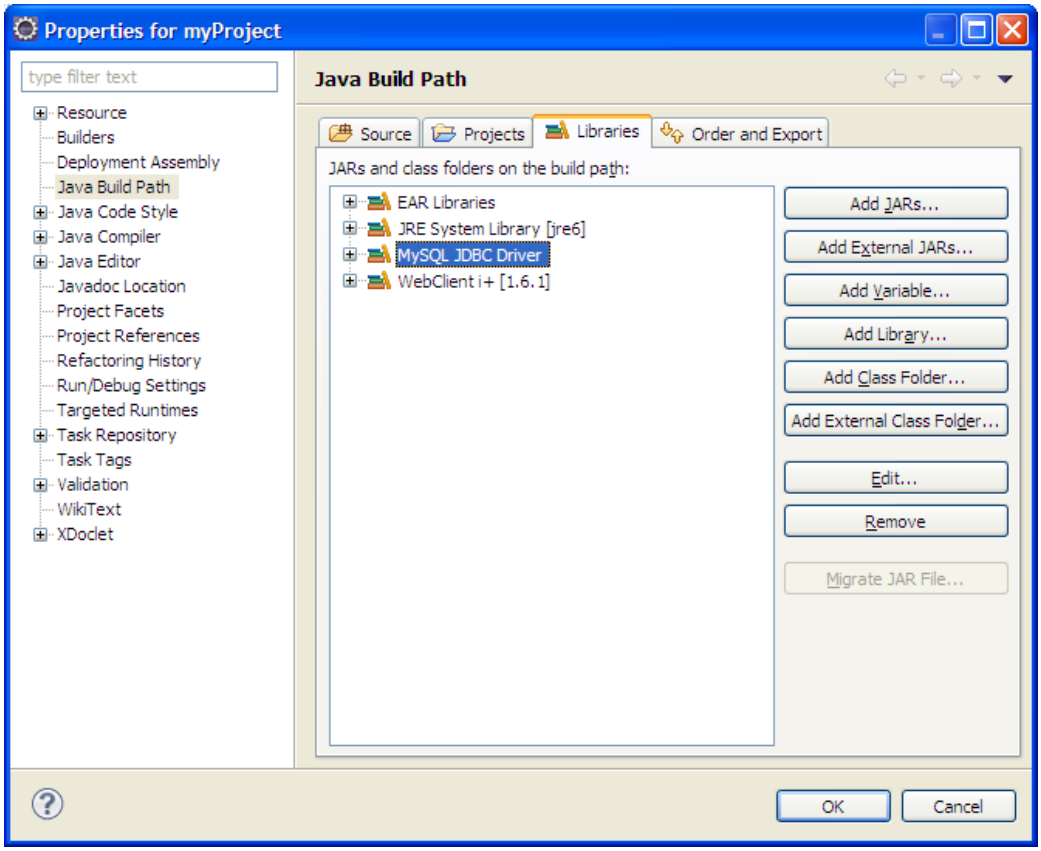
```
Environment.Default.Driver=com.mysql.jdbc.Driver
Environment.Default.DataSource=jdbc:mysql://localhost:portnumb
er/yourdatabasename
Environment.Default.User=username
Environment.Default.Password=password
```

b. Add JDBC driver to your Java Project. Right-click on the java project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add Library…** → Select **Connectivity Driver Definition** → Click **Next** >



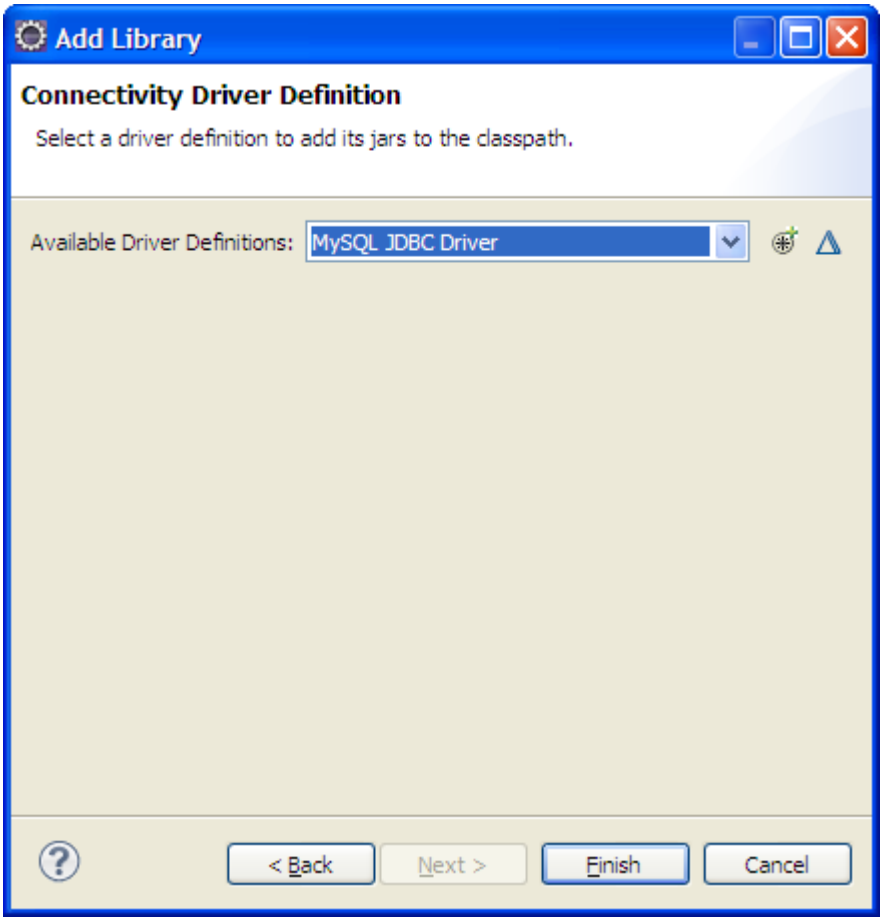
c. Click on **Add New Driver Definition** (an icon beside Available Driver Definitions combo box).
d. On the Name/Type tab select **MySQL JDBC Driver** according to your MySQL version
e. Open **JAR List** tab, click **Clear All** then click **Add JAR/Zip…** to select mysql-connector-java-5.1.13-bin.jar from your mysql JDBC installed directory.
f. Open **Properties** tab, enter your connection URL, database name and password according to obclient.properties
g. Click **Finish**.

Appendix B: JDBC Driver Configuration



i. Add the JDBC drive to your web project.
Right-click on your web project → Select **Properties** → Select **Java Build Path** → Open **Libraries** tab → Click **Add Library…** → **Select Connectivity Driver Definition** → Click **Next** > Select **MySQL JDBC Driver** from Available Driver Definitions → Click **Finish**.

Appendix B: JDBC Driver Configuration



Appendix C: Licensing

WebClient Live Installer includes development license for 10 users. If you do not have WebClient Live installed, comment out `licensingserver.licensefile=XXX`. It will automatically activate the development license for 10 users. Trial generator license is only valid for 30 days. For more information about licensing, please contact John Rhodes jdrhodes@cmfirsttech.com.

Appendix D: CM WebClient Support

Go to CM First Support Websites to report any issues and to request temporary license. Please include any log file and screenshots with your ticket. You need to be a registered user to use the site. Our support staff will contact you as soon as possible. It can sometimes take us up to 48 hours to respond because the most of our issues require research to resolve, though we make every effort to get back to your as quickly as possible. Most queries are answered within two business day.

Appendix E: CM WebClient Mobile

CM WebClient Mobile is free to download at CM First Support Website.

We recommend our CM WebClient users to create a separate workspace for their mobile development.

For more information about CM WebClient Mobile support, please contact support@cmfirstgroup.com.